UNIVERSIDAD AUTONOMA DE MADRID

ESCUELA POLITECNICA SUPERIOR

Grado en Ingeniería en Tecnologías y Servicios de Telecomunicación

TRABAJO FIN DE GRADO

Desarrollo de un recolector de datos de monitorización para dispositivos de red basado en protocolos estandarizados

Borja Pascual Rueda
Tutor: Lluís Gifre Renom
Ponente: Jorge Enrique López de Vergara Méndez
Mayo 2018

Desarrollo de un recolector de datos de monitorización para dispositivos de red basado en protocolos estandarizados

AUTOR: Borja Pascual Rueda

TUTOR: Lluís Gifre Renom

PONENTE: Jorge Enrique López de Vergara Méndez

High Performance Computing and Networking Research Group (HPCN)

Dpto. Tecnología Electrónica y de las Comunicaciones

Escuela Politécnica Superior

Universidad Autónoma de Madrid

Mayo de 2018

Resumen (castellano)

El futuro incremento de demanda en ancho de banda y estabilidad en las redes de computadores, debido a la aparición y popularización de servicios como el video bajo demanda o aplicaciones basada en IoT (Internet of Things, Internet de las cosas), motiva que las compañías operadoras de redes de comunicaciones optimicen el uso de sus redes y ofrezcan un servicio de mejor calidad y precio a los usuarios, tanto domésticos como empresariales, para respetar los acuerdos de servicio pactados con ellos. Para poder respetar los acuerdos de calidad de servicio y disponibilidad es necesario monitorizar la red con el objetivo de conocer que está ocurriendo en la misma y si esos eventos pueden afectar a la calidad o disponibilidad de dicho servicio.

La motivación de este Trabajo Final de Grado surge debido a la heterogeneidad existente en lo relativo al soporte de distintos protocolos de monitorización por parte de los distintos dispositivos de red. Por tanto, el objetivo de este trabajo será diseñar y desarrollar un sistema capaz de unificar las interfaces de monitorización de distintos dispositivos de red. De esta forma, se podrán recolectar los datos de monitorización que estos reportan y traducirlos a un formato común permitiendo que un sistema de gestión de red pueda obtener dichos datos a través de una única interfaz. Una característica importante del sistema será su facilidad de extensión para soportar nuevos protocolos y tipos de datos monitorizados.

En este proyecto haremos uso del protocolo SNMP (Simple Network Management Protocol, Protocolo Simple de Gestión de Red) para obtener datos de monitorización e utilizaremos distintas MIBs (Management Information Base, Base de Información de Gestión) para obtener dicha información; entre otras, se empleará la MIB RMON (Remote Network Monitoring, Monitorización Remota de Red). Usaremos un protocolo basado en gRPC (Google Remote Procedure Call, Llamada a Procedimiento Remoto de Google) para que un sistema superior, como puede ser una plataforma de monitorización, pueda comunicarse con nuestro sistema y sea capaz de mandarnos peticiones. Para transmitir la información solicitada por parte de dicha plataforma de monitorización, utilizaremos el protocolo estandarizado IPFIX (Internet Protocol Flow Information Export, Exportación de Información de Flujos del Protocolo de Internet).

Por último, se realizarán pruebas del sistema enviándole comandos a través de una interfaz de línea de comandos que emule la mencionada plataforma de monitorización. De esta forma validaremos el correcto funcionamiento del sistema.

Palabras clave (castellano)

Recolección de tráfico de red, plataformas de monitorización, calidad de servicio.

Abstract (English)

The future increase in demand for bandwidth and stability in computers networks, due to the emergence and popularization of services such as video on demand or applications based on IoT (Internet of Things), motivates computer networks' operators to optimize the use of their networks and offer a service of better quality and price to both domestic and business users to respect the service agreements approved with them. In order to be able to respect service quality and availability agreements, it´s of paramount importance to monitor the network in order to know what is happening on it and if those events may affect the quality or availability of service.

The motivation of this end-of-degree project arises due to the existing heterogeneity on the set of monitoring protocols supported by the different network devices available. Therefore, the objective of this work will be to design and develop a system able of unifying the monitoring interfaces of different network devices; thus enabling to collect monitoring data reported by them and translating them into a common format allowing a network management system can obtain such data through a single interface. An important feature of the system will be its ease of extension to support new protocols and monitored data types.

In this project we will use the SNMP (Simple Network Management Protocol) protocol to obtain monitoring data and we will use different MIBs (Management Information Base) to obtain this information; among others, the RMON (Remote Network Monitoring) MIB will be used. We will use gRPC (Google Remote Procedure Call)-based protocol to enable an upper-layer management system, such as a monitoring platform, to communicate with our system and send requests to us. To transmit the information requested by the aforementioned monitoring platform, we will use the standardized IPFIX (Internet Protocol Flow Information Export) protocol.

Finally, system tests will be carried out by sending commands through a command line interface that emulates the aforementioned monitoring platform. In this way we will validate the correct functionality of the system.

Keywords (English)

Network traffic collection, monitoring platforms, quality of service.

Agradecimientos

Me gustaría agradecer a mis padres la oportunidad de estudiar esta carrera y las facilidades que me han dado para que pudiese realizarla de la forma más amena posible, en especial agradezco a mi madre su apoyo incondicional todos estos años, que en ocasiones han sido muy estresantes y ahí estaba para tranquilizarme. También agradezco el apoyo y confianza que tenía mi abuela en mí, espero que allí donde estés te sientas orgullosa de tu nieto. No me olvido de agradecer la oportunidad que mi tutor, Lluís, me ha brindado a la hora de realizar este trabajo y todo el apoyo e interés que he tenido por parte de él. Por último, agradecer a mis amigos y a Violeta por aguantarme, incluso en esos días malos.

ÍNDICE DE CONTENIDOS

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

Glosario

API	Application Programming Interface
ASN.1	Abstract Syntax Notation One
BER	Basic Encoding Rules
IP	Internet Protocol
IPFIX	IP Flow Information Export
MAC	Media Access Control
MAC	Medium Access Control
MIB	Management Information Base
NMS	Network Management Stations
OID	Object Identifier
RMON	Remote Network Monitoring
RPC	Remote Procedure Call
SMI	Structue of Management Information
SNMP	Simple Network Management Protocol
UDP	User Datagram Protocol

1 Introducción

En este capítulo motivaremos la necesidad de realizar el proyecto que se ha abordado en este Trabajo Final de Grado (TFG), detallaremos el objetivo y sub-objetivos en que este se divide, presentaremos la planificación del proyecto y, finalmente, resumiremos la organización de esta memoria.

1.1 Motivación

En los últimos años, Internet está sufriendo un crecimiento exponencial en cantidad de dispositivos y tráfico transportado; según Cisco el tráfico IP anual en 2016 fue de 1.1 ZB (ZetaBytes, 1 ZB = 10^9 TeraBytes); para 2021 se espera que esa cifra aumente hasta 3.3 ZB[1]. La aparición de servicios como el video bajo demanda o las aplicaciones basadas en IoT (Internet of Things, internet de las cosas) propician un incremento en la demanda de ancho de banda y estabilidad nunca antes vista ni esperada.

El mencionado incremento en ancho de banda y estabilidad motiva que las compañías operadoras de redes de comunicaciones optimicen el uso de sus redes y ofrezcan un servicio de mejor calidad y precio a los usuarios, tanto para los domésticos como empresariales, para respetar los acuerdos de servicio pactados con sus clientes. Para poder respetar los acuerdos de calidad de servicio y disponibilidad es necesario monitorizar la red con el objetivo de conocer que está ocurriendo en la misma y si dichos cambios pueden afectar a la calidad o disponibilidad del servicio.

Un problema recurrente en la monitorización de redes es la heterogeneidad en los dispositivos de red; dos dispositivos, incluso del mismo fabricante, pueden no reportar los datos de monitorización de la misma manera. A pesar de que la mayoría de dispositivos soportan protocolos estandarizados, por ejemplo SNMP (Simple Network Management Protocol, Protocolo Simple de Administración de Red), como medio para transmitir datos de monitorización, pueden tener implementadas distintas estructuras de organización de los datos, llamadas MIBs (Management Information Base, Base de información de gestión). Por ejemplo, en la MIB RMON (Remote Network Monitoring, Monitorización Remota de Red), la propia RFC 2819 [2] especifica que el dispositivo sólo debe implementar los grupos que utilice y con los que tenga dependencias. Incluso, existen MIBS propietarias definidas por los distintos fabricantes de dispositivos.

Ante esta situación, crear un sistema que sea capaz de comunicarse con los distintos dispositivos y sea capaz de traducir la información a un lenguaje común, permite que sistemas de gestión de red (fuera del alcance de este TFG) sean capaces de operar gran variedad de dispositivos y utilizar los datos reportados de una manera fácil y ágil sin la necesidad de conocer las particularidades de cada dispositivo. Entonces, dichos sistemas de gestión podrán tomar decisiones a partir de los datos obtenidos para prevenir posibles problemas antes incluso de que se produzcan, ahorrando grandes cantidades de dinero a las operadoras.

1.2 Objetivos

Este TFG tiene como objetivo diseñar y desarrollar un sistema capaz de unificar las interfaces de monitorización de distintos dispositivos de red, para poder recolectar los datos de monitorización que estos reportan y traducirlos a un formato común permitiendo

que un sistema de gestión de red pueda obtener dichos datos a través de una única interfaz. Para alcanzar este objetivo principal, el proyecto se ha organizado en una serie de sub-objetivo que se resumen seguidamente:

1. **Estudio de protocolos de monitorización:** Este sub-objetivo tiene por misión investigar los distintos protocolos de monitorización y las operaciones que estos ofrecen, así como la posibilidad de implementar dichos protocolos y operaciones en Python.
2. **Implementación del sistema:** En este sub-objetivo se implementarán los distintos módulos necesarios para el correcto funcionamiento del sistema. Además, se llevará a cabo la integración de los mismos.
3. **Pruebas:** Tras realizar la implementación, en este sub-objetivo demostraremos, mediante una serie de casos de prueba, el correcto funcionamiento del sistema.

1.3 Fases del proyecto

Las distintas tareas del proyecto, se ilustran mediante un diagrama de Gantt en la Figura 1-1, y se resumen en la Tabla 1-1. Dichas tareas, se han agrupado en cuatro grandes grupos que se describen a continuación. El primero, denominado "Monitorización", se centra en la investigación de los protocolos SNMP y RMON, así como de las MIBs y operaciones a utilizar en este proyecto.

En el segundo grupo de tareas, denominado "Implementación del Sistema", se ha invertido la mayor parte del tiempo de este proyecto. En este grupo se engloban todas las tareas de implementación, y se realiza la integración de las mismas. También contempla la implementación de la interfaz de control que nos permitirá recibir comandos de sistemas de gestión de red y realizar las distintas pruebas de nuestro sistema.

En el tercer grupo de tareas, denominado "Pruebas", se implementa el módulo VTY (Virtual Teletype, emulador terminal) y se usa para probar el sistema. El módulo VTY emula un sistema de gestión de red que envía comandos a través de la interfaz de control a nuestro sistema para poder realizar las pruebas y correcciones pertinentes.

Por último, reservamos parte del tiempo de realización del proyecto para preparar tanto la memoria como la presentación de la misma. La tarea de preparación de memoria se ha intentado realizar en paralelo con las fases de implementación y pruebas con la finalidad de agilizar la realización de la misma.

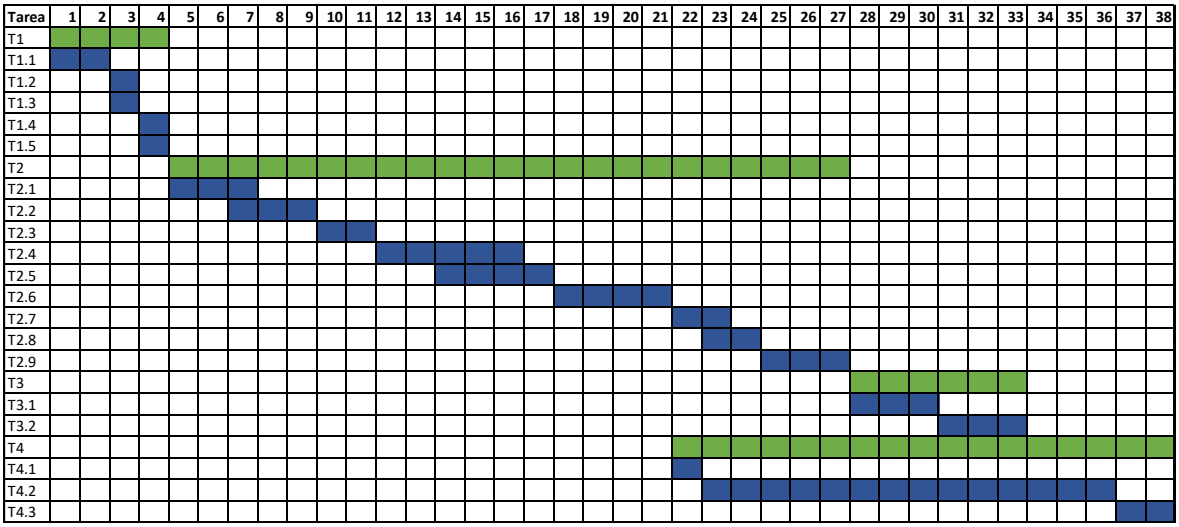


Figura 1-1 – Diagrama Gantt del proyecto

Tabla 1-1 – Tareas del proyecto

Tarea	Actividad	Duración
T1	Monitorización	4
T1.1	Investigación sobre el protocolo SNMP	2
T1.2	Investigación sobre la MIB RMON	1
T1.3	Investigación sobre las operaciones de SNMP	1
T1.4	Investigación las MIBs del conmutador Dlink	1
T1.5	Despliegue en entorno GNS3	1
T2	Implementación del sistema	23
T2.1	Investigación PySNMP	3
T2.2	Implementación de las operaciones de SNMP sobre las MIBs del dispositivo	3
T2.3	Implementación de los Samplers	2
T2.4	Implementación del módulo SNMP	5
T2.5	Implementación del módulo RMON	4
T2.6	Implementación del módulo Manager	4
T2.7	Investigación del lenguaje de gRPC	2
T2.8	Implementación de la interfaz de control basada en gRPC	2
T2.9	Integración de los módulos	3
T3	Pruebas	6
T3.1	Implementación de la terminal VTY + GRPC	3
T3.2	Pruebas, correcciones y ampliaciones	3
T4	Presentación y Memoria	17
T4.1	Preparación de la tabla de contenidos	1
T4.2	Preparación de la memoria	14
T4.3	Preparación de la presentación	2

1.4 Organización de la memoria

La memoria consta de los siguientes capítulos:

- En el capítulo 2, titulado **Estado del arte**, se describirán los conceptos más importantes y protocolos que se han utilizado para implementar el sistema recolector.
- En el capítulo 3, titulado **Diseño y Desarrollo**, se detalla el diseño que se ha realizado para desarrollar el sistema recolector y las particularidades relacionadas con el desarrollo e implementación del mismo.
- En el capítulo 4, titulado **Integración, pruebas y resultados**, se plantean un conjunto de casos de prueba que nos permitirán validar el correcto funcionamiento del sistema recolector, así como su posible integración en un sistema mayor de gestión de red.
- Finalmente, en el capítulo 5 titulado **Conclusiones y trabajo futuro**, resumimos las conclusiones que se derivan de la realización de este proyecto, así como las posibilidades de desarrollo y mejora que puede tener este sistema de cara al futuro.

2 Estado del arte

En este capítulo, hablaremos del estado del arte relevante para este proyecto. Introduciremos los conceptos más importantes de una red de comunicaciones y protocolos, para después adentrarnos en conceptos propios de monitorización de redes y los distintos protocolos utilizados para desarrollar el sistema.

2.1 Redes de computadores

Una red de computadores consiste en un conjunto de equipos o sistemas terminales conectados entre sí, permitiéndoles así compartir e intercambiar datos. Dichos sistemas terminales se conectan entre sí mediante un conjunto de enlaces de comunicaciones y dispositivos de intercambio de paquetes [3].

Cada dispositivo de red se encarga de tomar un paquete que llega a través de uno de sus puertos conectado a un enlace y lo reenvía a través de otro de sus puertos, que también estará conectado a otro enlace. Los enlaces de comunicaciones son cada uno de los caminos por el que podemos transmitir información; los enlaces físicos más comunes son los cables eléctricos, de fibra óptica y los radio enlaces. Los dos tipos más utilizados de dispositivos de red son los enrutadores (routers en inglés) y los conmutadores (switches en inglés).

La función de un conmutador de red es recibir las tramas de la capa de enlace entrantes y reenviarlas a los enlaces de salida apropiados. Entonces, los conmutadores permiten que los equipos en una misma red se comuniquen entre sí [4]. El propio conmutador es transparente para los distintos equipos de la red y realiza el reenvío de los paquetes utilizando las direcciones de la capa de enlace o direcciones MAC (Medium Access Control, control de acceso al medio), que son identificadores únicos para cada dispositivo físico de una red [3]. Existen dos tipos de conmutadores:

- **Conmutadores no gestionados:** No pueden ser configurados ni monitorizados.
- **Conmutadores gestionados:** Pueden ser monitorizados y configurados de manera local o remota permitiendo así controlar el tráfico de la red.

Los enrutadores son dispositivos de red que encaminan paquetes entre redes distintas, esto es, permiten conectar varias redes entre sí. Actúa como despachador extrayendo determinados campos de los paquetes que se envían a través de la red para elegir la mejor ruta para que los datos lleguen a su destino; para realizar el reenvío de los paquetes hace uso de las direcciones de la capa de red o direcciones IP (Internet Protocol, Protocolo de Internet), que pueden ser reconfiguradas para cada dispositivo [4].

2.2 Protocolos

Además de los dispositivos de red, es necesario definir protocolos para que dichos dispositivos se entiendan entre ellos. Un protocolo define el formato y el orden de los mensajes intercambiados entre dos o más entidades que se comunican, así como las acciones tomadas al producirse la transmisión y/o recepción de un mensaje u otro suceso [3].

Los distintos protocolos de red se organizan en capas. Un protocolo en una capa determinada sólo puede comunicarse con el protocolo inmediatamente superior o inferior, si los hubiera, pero no con el resto. En estas capas distinguimos las siguientes:

- la capa de aplicación, que es donde residen las aplicaciones de red y sus protocolos, por ejemplo, HTTP (HyperText Transfer Protocol, Protocolo de transferencia de hipertexto), FTP (File Transfer Protocol, Protocolo de transferencia de archivos).
- la capa de transporte que es la encargada de transportar los mensajes de la capa de aplicación. Entre los protocolos más destacados, encontramos UDP (User Datagram Protocol, Protocolo de Datagramas de Usuario) y TCP (Transmission Control Protocol, Protocolo de control de transmisión).
- la capa de red que es la encargada de transportar los paquetes de red, conocidos como datagramas, de un equipo a otro. En esta capa, cabe destacar el protocolo IP.
- la capa de enlace, que es la encargada de trasladar los paquetes de un nodo de red físico a otro adyacente en la red. El protocolo posiblemente más importante en esta capa es Ethernet.
- la capa física, que se encargaría del procesado de la señal y de transportar los bits individualmente de un dispositivo de red o equipo al siguiente.

El protocolo de capa de red IP, proporciona los medios necesarios para la transmisión de bloques de datos llamados datagramas desde el origen al destino. El origen y el destino son equipos de red identificados por direcciones de longitud fija, llamadas direcciones IP, que están compuestas por 4 bytes. Normalmente, estas direcciones se representan separadas por puntos e identifican unívocamente un dispositivo en una red. También provee de mecanismos de fragmentación y re-ensamblaje de datagramas, si fuera necesario, para poder transmitir los paquetes a través de redes que soporten distintas longitudes máximas de paquetes [5].

El protocolo de capa de transporte UDP tiene por objetivo proporcionar un mecanismo simple para que los programas envíen mensajes a otros programas. El protocolo se orienta a transacciones y tanto la entrega como la protección ante duplicados no se garantiza. Dado que no existe una fase de establecimiento de la conexión entre las entidades de la capa de transporte emisora y receptora previa al envío del segmento, se dice que es un protocolo sin conexión [6]. Asume que el protocolo IP se utiliza como protocolo de la capa inferior. Con el objetivo de identificar y multiplexar múltiples conexiones o flujos de datos entre un mismo par de dispositivos, las distintas aplicaciones y servicios se etiquetan con un identificador numérico llamado puerto.

El protocolo de capa de transporte TCP, a diferencia de UDP, está orientado a conexiones, diseñado para ser utilizado como protocolo fiable entre equipos en una red de conmutación de paquetes y en sistemas conectados a esa red. El protocolo, por tanto, garantiza la entrega de los datos en destino sin errores y en el mismo orden que se transmitieron [7]. Al ser un protocolo orientado a conexiones, se definen tres etapas claras en la operación de TCP: *i*) *establecimiento de conexión*, en la que los extremos de la conexión acuerdan los parámetros de la misma, *ii*) *transferencia de datos*, que consiste en el intercambio de mensajes una vez la conexión está establecida, y *iii*) *final de la conexión*, en el que los extremos se desconectan entre ellos. Al igual que UDP, TCP emplea puertos

para multiplexar diferentes conexiones o flujos de datos entre un mismo par de dispositivos.

2.3 Monitorización de redes

La monitorización de redes consiste en inspeccionar y analizar el tráfico que circula por dicha red con el objetivo de medir su rendimiento, detectar errores e, incluso, predecirlos para tratar de prevenir posibles problemas mayores.

En la monitorización de redes son necesarios varios componentes; por un lado, dentro del dispositivo de red, o como dispositivo aparte conectado a la red, tenemos las *sondas* que capturan los paquetes de la red, pudiéndolos agregar y contabilizar, si es necesario, dejando estos datos disponibles para poder ser solicitados desde el exterior del dispositivo a través de un módulo exportador. También es necesario un sistema de recolección capaz de conectar con los distintos dispositivos de red o sondas y obtener las medidas. Por último, se necesita un protocolo común entre ambos que permita configurar los distintos componentes que se encargan de la captura de esas muestras y que el sistema de recolección pueda recuperar las muestras enviadas por el exportador.

Algunos de los protocolos de monitorización de redes estandarizados y más usados en la actualidad son SNMP, incluyendo las MIBs estandarizadas como las de RMON, e IPFIX. Dichos protocolos serán empleados por el sistema que se desarrollará en este proyecto para interactuar con los dispositivos de red y para exportar los datos de monitorización recolectados a sistemas de capas superiores.

2.3.1 SNMP

El protocolo SNMP (Simple Network Management Protocol, Protocolo Simple de Administración de Red) es un protocolo de la capa de aplicación que facilita el intercambio de información de administración entre dispositivos de red y equipos de gestión de red [8]. El núcleo de SNMP incluye un conjunto de operaciones que permiten a los gestores de red cambiar el estado de algún dispositivo basado en SNMP [9].

En la Figura 2-1 podemos observar los componentes principales de la arquitectura SNMP. El gestor (*Manager*), es un servidor que se encarga de ejecutar algún tipo de aplicación capaz de administrar una red, son también conocidos como NMS (Network Management System, Sistema administrador de red). Para que el gestor pueda administrar los dispositivos de red, se necesita tener agentes, que son pequeños módulos software que se ejecutan en los distintos dispositivos gestionados, tales como servidores y dispositivos de red, entre otros; y provee la interfaz para que el gestor interactúe vía SNMP con los dispositivos.

Existen dos mecanismos de comunicación entre el gestor y los agentes: *encuesta* (*poll*) que consiste en preguntar a un agente por una información y esperar a su respuesta, y *notificación* (*trap*) que permite al agente mandar un mensaje de forma asíncrona al NMS, por ejemplo, para comunicarle algún evento originado en el propio dispositivo. Notar que SNMP es un protocolo de capa de aplicación que normalmente funciona sobre UDP, aunque puede ser configurado para operar sobre TCP también. Utiliza dos puertos destino diferentes para diferenciar entre mensajes enviados por encuesta, que usa el puerto 161, y por notificación que usa el puerto 162.

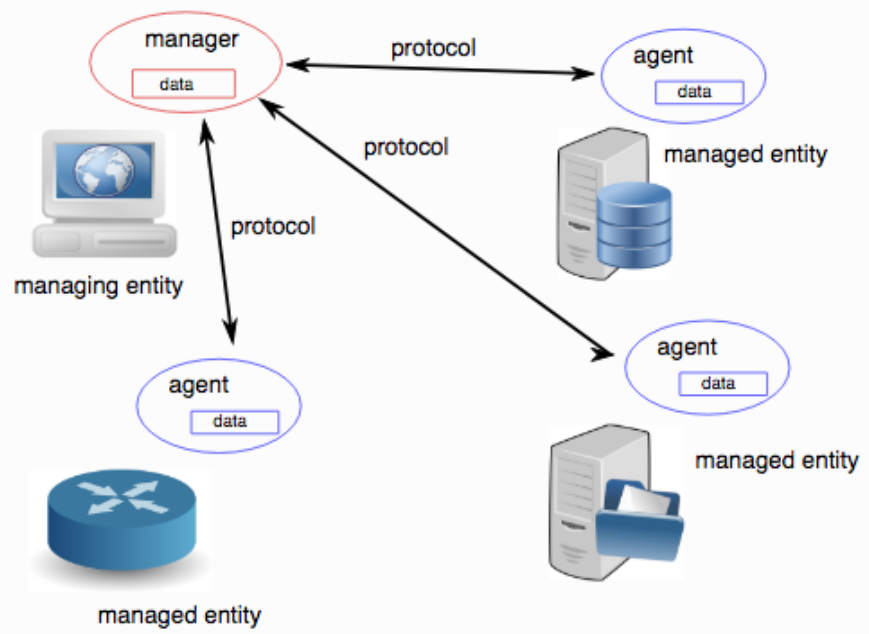


Figura 2-1 – Esquema SNMP. Fuente:[20]

2.3.1.1 MIB

Los objetos gestionados son accedidos a través de un almacén de información virtual, denominado MIB (Management Information Base, Base de información de gestión) [10]. Los mensajes que intercambian el gestor con los agentes, se especifican mediante el lenguaje de definición de datos llamado SMI (Structure of Management Information, Estructura de información de gestión) [11]. SMI provee mecanismos para definir la estructura de los objetos gestionados y su comportamiento, así como los objetos que componen las MIBs.

A su vez, se define la notación de cada uno de los objetos de las MIB empleando ASN.1 (Abstract Syntax Notation One); ASN.1 es un estándar que define una notación formal para describir las representaciones de los datos transmitidos por protocolos de comunicaciones, independientemente de la implementación, del lenguaje y la representación física de esos datos [12].

Cada objeto gestionado tiene un nombre, una sintaxis y una codificación. El nombre o OID (object identifier, identificador de objeto) identifica unívocamente un objeto. Los objetos gestionados se organizan jerárquicamente en forma de árbol, como se puede ver en el ejemplo de la Figura 2-2. Cada OID se compone de una serie de valores enteros basados en los nodos del árbol y separados por puntos. Se puede usar la secuencia de números o la secuencia de nombres que representan esos números para identificar cada objeto gestionado.

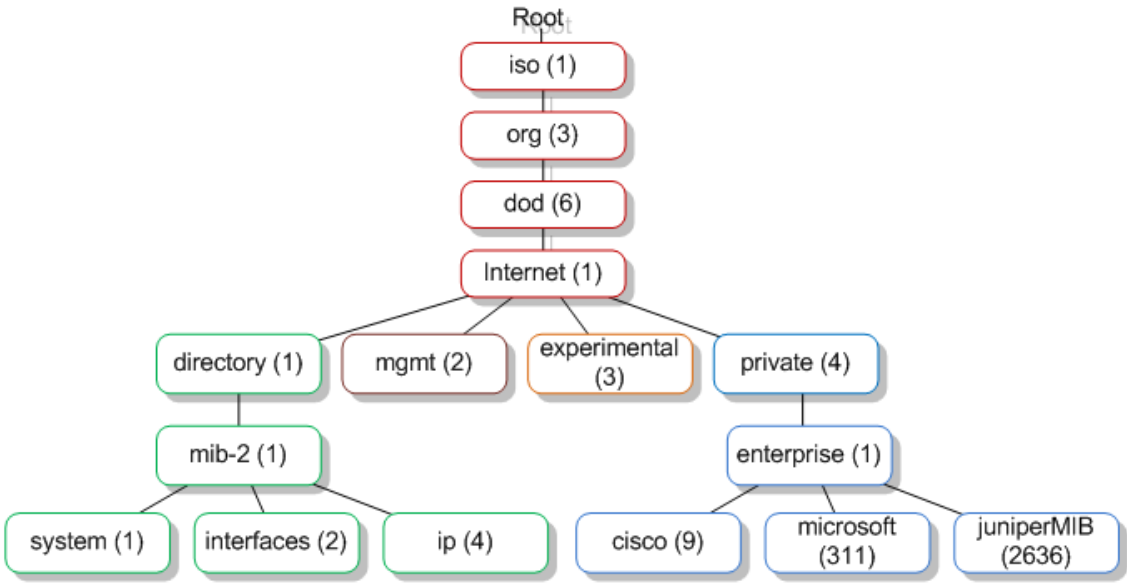


Figura 2-2 – Ejemplo Árbol OID. Fuente: [13]

Una instancia única de un objeto administrado se codifica en una cadena de octetos utilizando BER (Basic Encoding Rules) [12]. BER define cómo los objetos son codificados y decodificados para así poder ser transmitidos a través de cualquier medio de transporte y entre cualquier par de sistemas.

Como ejemplo de definición de un objeto gestionable, en la Tabla 2-1, se describe el objeto `etherStatsPkts` perteneciente al grupo `Statistics` de la MIB RMON (que se describirá en la sección 2.3.1.3):

Tabla 2-1 – Ejemplo Notación ASN.1

Notación ASN.1	{iso(1) identified-organization(3) dod(6) internet(1) mgmt(2) mib-2(1) rmon(16) statistics(1) etherStatsTable(1) etherStatsEntry(1) etherStatsPkts(5)}
Notación mediante puntos	1.3.6.1.2.1.16.1.1.1.5

Descripción	`etherStatsPkts OBJECT-TYPE`
	`SYNTAX Counter`
	`ACCESS read-only`
	`STATUS mandatory`
	`DESCRIPTION`
	`"The total number of packets (including bad packets, broadcast packets, and multicast packets) received."`

2.3.1.2 Operaciones

SNMP soporta diversas operaciones; cada una de ellas permite realizar un tipo de pregunta por parte de un gestor a un agente. Las operaciones son:

- **GetRequest/GetResponse:** El gestor inicia la solicitud al agente de un dato concreto enviando un mensaje *GetRequest*. El agente recibe la solicitud y, si tiene

éxito en la recopilación de la información solicitada, responde de vuelta al gestor con un mensaje *GetResponse*.

- **GetNext**: El mensaje *GetNext* permite enviar una secuencia de comandos para obtener un grupo de valores de una MIB. Para cada objeto MIB que solicitamos, se generan una solicitud *GetNext* y un mensaje de respuesta *GetResponse*.
- **GetBulk**: El mensaje *GetBulk* permite obtener una sección de una tabla de una vez; tiene dos parámetros: *nonrepeaters*, que indica que los primeros *N* objetos se pueden recuperar con una operación *GetNext*; y *maxreptition*, que especifica al agente que intente realizar *M* operaciones *GetNext* para recuperar los objetos restantes.
- **Set**: El mensaje *Set* permite cambiar el valor de un objeto gestionado o crear una nueva fila en una tabla. El agente responde si hay un problema con *GetResponse*.

2.3.1.3 RMON

La MIB RMON (Remote Network Monitoring, Monitorización Remota de Red) permite que varios dispositivos de red y un sistema de gestión de red intercambien datos de monitorización. Los monitores o sondas son instrumentos que tienen como objetivo capturar tráfico y monitorizar una red [2]. Entre los principales objetivos de RMON encontramos los siguientes:

i) La capacidad de operar sin conexión, permitiendo que la sonda se configure para realizar diagnósticos y recopilaciones de estadísticas de manera continua, incluso cuando la comunicación con el gestor puede no ser posible o eficiente; en este último caso la sonda puede intentar notificar al NMS cuando ocurre una condición excepcional.

ii) Otro objetivo a tener en cuenta es monitorización proactiva; el monitor está siempre disponible ante cualquier fallo, incluso puede notificar al gestor el fallo y almacenar la información recopilada sobre el mismo.

iii) El monitor se puede configurar para reconocer las condiciones que van a llevar a un error y cuando se produzca alguna de esas condiciones generar un evento de notificación al gestor, lo que en la RFC se denomina “detección y reporte de problema”.

iv) Por último, se permite que una organización pueda tener varios NMS.

Dado que la cantidad de información que puede gestionarse relacionada con las capturas puede ser elevada, la MIB RMON define los siguientes grupos de objetos para organizar la información de forma más clara:

- *Statistics*: Contiene estadísticas medidas por la sonda para cada interfaz de red supervisada por este dispositivo.
- *History Control*: Controla el muestreo periódico de datos estadísticos.
- *History:* Guarda muestras estadísticas periódicas.
- *Alarms*: Define un conjunto de umbrales para monitorizar el rendimiento de la red. Si un umbral se cruza en una dirección concreta generará un evento. *Alarms* necesita *Event*.
- *Host*: Contiene estadísticas asociadas con la dirección MAC de cada equipo descubierto en la red.
- *HostTopN*: Muestra *N* primeros equipos de la red que maximizan un criterio especificado de ordenación. *HosTopN* necesita el grupo *Host*.

- *Matrix*: Recopila información sobre el tráfico entre dos equipos dentro de una subred.
- *Filter*: Permite que los paquetes que coincidan con un filtro dado puedan ser combinados formando una secuencia de datos que pueden capturarse o generar eventos.
- *Packet Capture*: Captura los paquetes de uno de los canales definidos en el grupo *Filter*. *Packet Capture* necesita el grupo *Filter*.
- *Event*: Controla la generación y notificación de eventos.

Existe un conjunto de grupos pertenecientes a la versión 2 de RMON que, dado que no hemos tenido acceso a ningún dispositivo que lo soportara y por motivos de espacio en la memoria, no se han incluido en esta memoria.

2.3.2 IPFIX

El protocolo IPFIX (IP Flow Information Export, Protocolo para la exportación de información de flujos de Internet) define cómo transmitir información sobre los flujos de datos en una red desde un proceso exportador a otro proceso recolector [14]. En el esquema general de IPFIX, podemos identificar, interno o adherido a un dispositivo de red dado, un módulo que se encarga de tomar medidas sobre los paquetes que circulan por el dispositivo y otro módulo, conocido como exportador (*Exporter* en inglés), que tiene como objetivo codificar y enviar el mensaje IPFIX que contiene los datos monitorizados. En el lado de recepción, tenemos un módulo, denominado recolector (*Collector* en inglés), que se encarga de recibir los mensajes IPFIX y decodificarlos.

Uno de los elementos esenciales en IPFIX es la plantilla (*template* en inglés), que permite flexibilizar el formato de registro permitiendo que el proceso de recolección procese mensajes IPFIX sin necesidad de conocer previamente la interpretación o formato de los registros enviados. Una *template* puede contener cualquier combinación de identificadores de campos, ya sean estandarizados por la IANA [15] o específicos de una empresa. Estas plantillas, deben enviarse previamente a cualquier registro de datos que las utilice, ya sea en el mismo mensaje, o en un mensaje anterior.

En la Figura 2-3, se muestra la estructura general de un mensaje IPFIX. Su cabecera contiene, entre otros, los campos de tiempo de exportación (*Export Time)*, que es la marca de tiempo, codificada como un valor entero de 32 bits, correspondiente al instante de tiempo en que el mensaje IPFIX deja el *Exporter*; y el campo identificador de dominio de observación (*Observation Domain Id*), que identifica al proceso exportador, de nuevo, codificada con un valor entero de 32 bits.

Después de la cabecera, siguen cero o más entradas de tipo "*Set*"; un "*Set*" es un conjunto o colección de registros llamados "*Records*" que tienen una estructura común, definida por la plantilla indicada en la cabecera del "*Set*". Destacamos dos "Set" utilizados en este proyecto, "*Template Set*", que es un tipo especial de "*Set*" que contiene un conjunto de "*Template Records*" y "*Data Set*", que contiene un conjunto de "*Data Records*". Cada "*Template Record*" se emplea para definir una plantilla y transporta los identificadores de los distintos elementos de información tanto de la IANA como de otra autoridad que conforman dicha plantilla. Por su parte, los "Data Record" contienen un conjunto de valores asociados a los distintos elementos cuyos identificadores se encuentran en la plantilla y contienen propiamente los valores de las muestras a recolectadas.

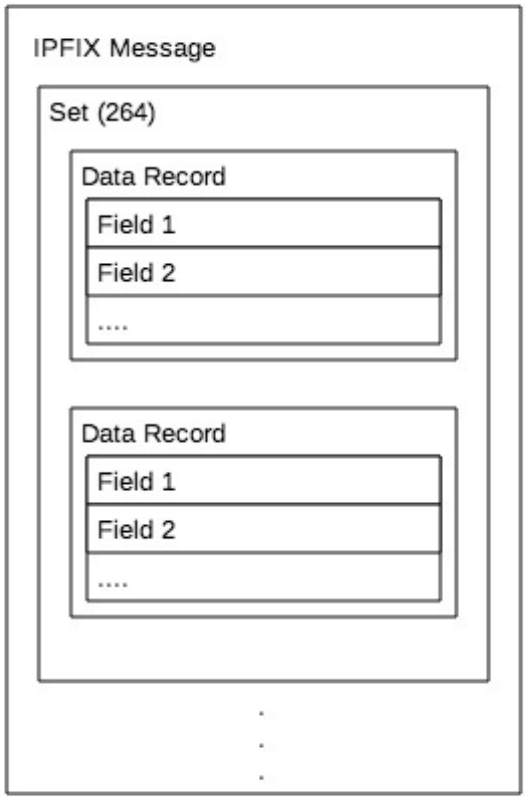


Figura 2-3 – Ejemplo Mensaje IPFIX

Los distintos componentes de un dispositivo de red, en IPFIX, se identifican mediante la asignación de puntos de observación (*Observation Points* en inglés), que son los puntos del dispositivo donde se conecta la sonda para extraer, por ejemplo, la información de los paquetes. Los puntos de observación normalmente se agrupan en dominios de observación (*Obsevation Domains* en inglés); estos últimos, normalmente, se mapean unívocamente con dispositivos físicos.

2.4 GRPC

gRPC es un nuevo lenguaje de definición de protocolos desarrollado por Google basado en el paradigma RPC (Remote Procedure Call, llamada a procedimientos remotos). Es de código abierto y multi-plataforma, con lo que se puede ejecutar en cualquier lugar. Permite a las aplicaciones cliente y servidor comunicarse de manera transparente facilitando así la creación de sistemas conectados [16]. La lista de RPCs que se definen en el protocolo, se agrupan en servicios, de forma que un mismo servicio ofrezca RPCs relacionadas entre sí.

gRPC define los protocolos mediante un fichero de texto plano con extensión ".proto", que codifica los servicios y la lista de RPCs que implementará cada uno de estos servicios. Este fichero es el que se utiliza para generar una clase representante (*Stub* en inglés) para el cliente, así como el módulo para el servidor.

En la Figura 2-4 se presenta un ejemplo de gRPC en el cual se conectan dos clientes diferentes a un mismo servidor. Una vez se define el protocolo de comunicaciones empleando gRPC, el generador de código de gRPC produce un *stub* que ofrece todas las RPCs definidas en el protocolo y que los clientes pueden usar para mandar peticiones al

servidor. Además, también se genera un módulo servidor de gRPC que se puede instanciar para ofrecer dichos servicios a los clientes.

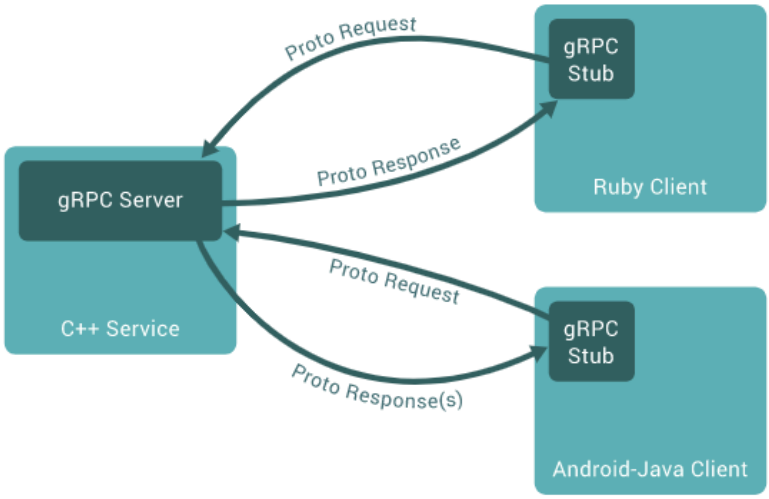


Figura 2-4 – Esquema gRPC Fuente: [16]

gRPC se basa en la idea de definir un servicio especificando los métodos que se puede llamar de forma remota con sus parámetros y tipos de retorno. En la Tabla 2-2 se detalla un ejemplo de servicio (líneas 1-6) que incluye 4 RPCs de distintos tipos de método de servicio, así como dos ejemplos de mensajes que contienen una cadena de texto cada uno como campos contenidos en el mensaje (líneas 7-12). Los cuatro tipos de método de servicio que se soportan son:

- **RPCs Unarios**, (línea 2 en Tabla 2-2): El cliente manda un único mensaje de petición al servidor y obtiene de vuelta otro mensaje individual de respuesta.
- **Transmisión a servidor RPCs**, (línea 3 en Tabla 2-2): El cliente manda un único mensaje de petición al servidor y obtiene de vuelta una secuencia de mensajes de respuesta.
- **Transmisión a cliente RPCs**, (línea 4 en Tabla 2-2): El cliente manda una secuencia de mensajes de petición al servidor y obtiene de vuelta un único mensaje de respuesta.
- **Transmisión Bidireccional RPCs**, (línea 5 en Tabla 2-2): El cliente manda una secuencia de mensajes de petición al servidor y obtiene una secuencia de mensajes de respuesta.

En el caso de las secuencias de mensajes de pregunta, las respuestas pueden empezar a enviarse inmediatamente después de recibir el primer mensaje de pregunta.

Tabla 2-2 – Ejemplo de gRPC

```
1  service Greeter{
2     rpc SayHello(HelloRequest) returns (HelloReply){}
3     rpc LotsOfReplies(HelloRequest) returns(stream HelloReply){}
4     rpc LotsOfGreetings(stream HelloRequest) returns(HelloReply) {}
5     rpc BidiChat(stream HelloRequest) returns (stream HelloReply){}
6  }
7  message HelloRequest {
```

```
 8     string name = 1;
 9  }
10  message HelloReply {
11     string message = 1;
12  }
```

Por defecto gRPC usa *protocol buffers*, una solución de Google para codificar mensajes estructurados de forma fácil; cada tipo de mensaje tiene un nombre y uno o varios campos numerados de manera única, tal como se muestra en la Tabla 2-2; cada campo tiene un nombre y un tipo de valor (entero, coma flotante, booleano, cadena de caracteres, etc). Además, un campo puede, a su vez, contener listas de valores, otro sub-mensaje, o incluso listas de sub-mensajes dándole mucha versatilidad a *protocol buffers* y por tanto a gRPC.

En las líneas 7-9 de la Tabla 2-2, se define un mensaje llamado "`HelloRequest`", que contiene un campo obligatorio de tipo cadena de caracteres, cuyo identificador es el 1, y se denomina "`name`". En las líneas 10-12, se define un mensaje llamado "`HelloReply`" que contiene el campo obligatorio denominado "`message`", de tipo cadena de caracteres, cuyo identificador es 1.

2.5 Conclusiones

En este capítulo, se ha repasado los conceptos importantes necesarios para entender este proyecto, se ha hablado de los componentes principales de una red, así como de los protocolos de capa de aplicación, de red y de transporte más utilizados.

También se ha introducido qué es la monitorización de redes así como los protocolos más conocidos; en particular SNMP, junto a su MIB RMON, y el protocolo IPFIX.

Por último, se ha hablado del lenguaje gRPC que permite definir de forma simple y rápida protocolos de comunicación entre aplicaciones cliente y servidor facilitando así la creación de sistemas conectados.

3 Diseño y Desarrollo

En este capítulo se describirá el diseño que se ha realizado para desarrollar el sistema de monitorización objeto de este proyecto y los detalles relacionados con el desarrollo e implementación del mismo.

En primer lugar, se ha realizado una selección de un dispositivo de ejemplo a monitorizar; seguidamente, se ha desarrollado un sistema genérico que pueda ser extendido tanto por desarrolladores como por vendedores de equipos de red. Finalmente, sobre este sistema, se ha hecho una especialización para poder monitorizar el dispositivo seleccionado.

3.1 Selección de dispositivos a monitorizar

Tras las primeras implementaciones de las operaciones SNMP a utilizar se estuvo estudiando la mejor manera de poner a prueba nuestro sistema. Para comenzar, probamos con un agente de simulación de SNMP que ofrecía PySNMP y servidores en línea [17], sin embargo, este no soportaba MIBs RMON.

Luego se intentó desplegar una red virtual de enrutadores utilizando la herramienta GNS3 [18] e imágenes de distintos enrutadores Cisco, como por ejemplo el Cisco C7200, pero todos ellos implementaban el soporte básico para RMON, que incluye solamente los grupos *alarms* y *events*; pero ninguno de ellos tenía soporte para los grupos de RMON que queríamos probar, por ejemplo *history* y *statistics*. El soporte para otros grupos debía solicitarse específicamente a Cisco y requería el pago de licencias.

Al final nos decantamos por un dispositivo físico que soportase RMON. En particular, se localizó un conmutador del fabricante DLink, concretamente el modelo DGS-3120-24PC [19], el cual ofrecía soporte para los grupos *history*, *statistics*, *alarms* y *events*, por lo que ha sido utilizado para las distintas pruebas del sistema desarrollado en este proyecto.

3.2 Arquitectura general

El sistema se ha diseñado de forma que sea extensible para que tanto desarrolladores como fabricantes de equipos puedan extenderlo con facilidad para soportar distintos dispositivos. Para ello, se ha definido, un sistema genérico al que se ha llamado *NodeController*. La arquitectura del *NodeController* y los módulos que lo conforman se muestran en la Figura 3-1. El diseño está pensado para poder ser extendido y soportar cualquier dispositivo de red que sea necesario.

El módulo principal de este sistema se denomina *Manager* y es el encargado de atender las peticiones que se soliciten desde un sistema de gestión de red a través del módulo *gRPC Server* y encaminarlas al módulo adecuado del sistema. También se encarga de enviar al recolector que se haya configurado las muestras recolectadas a través del módulo de exportación, llamado *IPFIX Exporter;* para el desarrollo de estos módulos se han utilizado las librerías pyIPFIX [26] y pyTools [27].

Además, el sistema permite desplegar un conjunto de agentes de monitorización, llamados *Monitoring Agents*, que implementan protocolos específicos. Cada uno de estos *Monitoring Agents*, a su vez, contiene un conjunto de muestreadores, llamados *Samplers*, que se encargan de configurar, si es necesario, y recolectar periódicamente la información

de monitorización de los dispositivos de red. La información recolectada se encapsula como muestra y se envía al *Manager*, quien la exportará al sistema recolector.

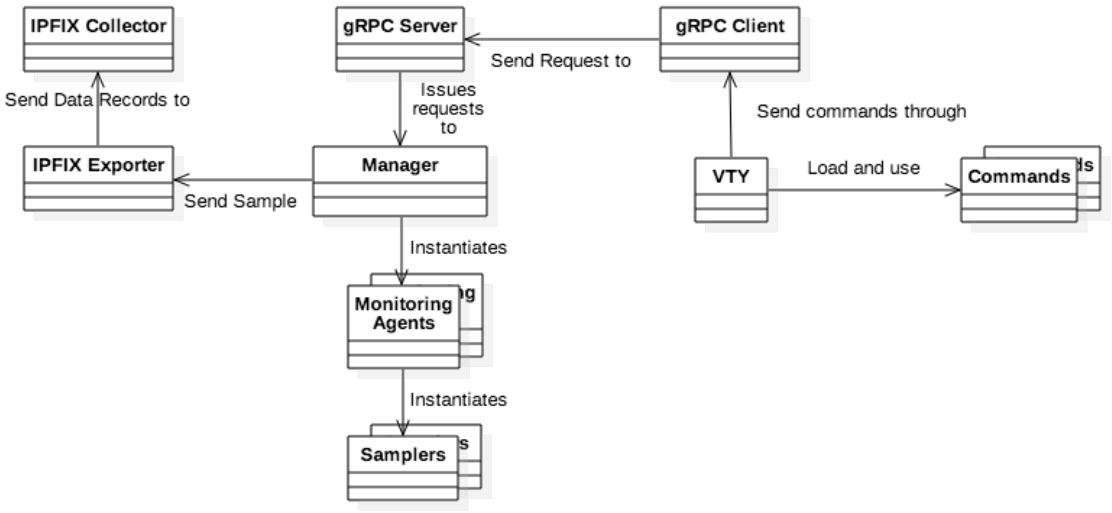


Figura 3-1 – Diseño simplificado del Node Controller

El *Manager* debe de ser capaz de registrar y configurar los distintos puntos de observación del dispositivo según las peticiones que reciba; en cada petición se detallará: el identificador del punto de observación y dominio de observación (en este caso el identificador del dispositivo de red), tipo de componente a monitorizar (este parámetro indicará, por ejemplo, interfaz, procesador, memoria, etc. y servirá para seleccionar el *Sampler* adecuado), el identificador del componente a monitorizar, el identificador de plantilla IPFIX que se usará para exportar las muestras y el periodo de monitorización. Las funciones principales de *Manager* son:

- La función **`CreateAgent`** se encarga de crear los distintos *Monitoring Agents* que emplea y configurarlos. Devuelve un error en caso de intentar crear un *Monitoring Agent* con un identificador que ya exista.
- La función **`ModifyAgent`** permite cambiar la configuración de un *Monitoring Agent*.
- La función **`ListAgent`** muestra una lista de *Monitoring Agents* configurados en el *NodeController*.
- Las funciones **`StartAgents`** y **`StopAgents`** tiene como función arrancar y parar, respectivamente, todos los *Monitoring Agents* que se hayan instanciado en el *NodeController*.
- La función **`DeleteAgent`** se encarga de borrar un agente existente y todos los puntos de observación asociados.
- La función **`CreateOP`** se encarga de crear y configurar los distintos *Observation Points*. La creación de un *Observation Point* implica directamente la instanciación de un *Sampler*. Devuelve un error en caso de intentar crear un *Observation Point* con un identificador que ya exista.
- La función **`ModifyOP`** modifica los parámetros de un *Observation Point* que se haya creado, permitiendo así reconfigurar su *Sampler* asociado. Devuelve un error en caso de intentar crear un Observation Point con un identificador que ya exista.

- La función **`DeleteOP`** se encarga de eliminar *Observation Point*, asociado al agente, con el identificador indicado.
- Las funciones **`EnableOP`** y **`disableOP`** permiten activar y desactivar, respectivamente, el *Sampler* asociado al *Observation Point* indicado.
- La función **`ListOP`** devuelve una lista con todos los *Observation Points* existentes.

Al arrancar, el *Manager* inicializa una instancia de *IPFIX Exporter* y lo configura con las plantillas que soporten los distintos *Samplers* que se hayan implementado en el *NodeController*. Para ello, se cargan las plantillas de un fichero de configuración, que se detalla en el Anexo A.C.2. Entonces, cuando un *Sampler* tiene una muestra lista para ser enviada, se la pasan a sus respectivos *Monitoring Agents* que a su vez la pasan al *Manager* para que cree el mensaje IPFIX adecuado con su correspondiente *Data Set* y *Data Records* según la plantilla que sea necesaria. Por último, se llama a la función sendMessage del *IPFIX Exporter* y este se encarga de enviar el mensaje IPFIX con la muestra.

3.3 Especialización para el dispositivo DLink DGS-3120-24PC

Como se ha comentado en la sección 3.1, la opción más viable a la hora de poder poner nuestro sistema ha sido utilizar un dispositivo físico, concretamente el DLink DGS-3120-24PC, por lo que hemos extendido nuestro sistema para soportar dicho Switch.

3.3.1 Esquema general

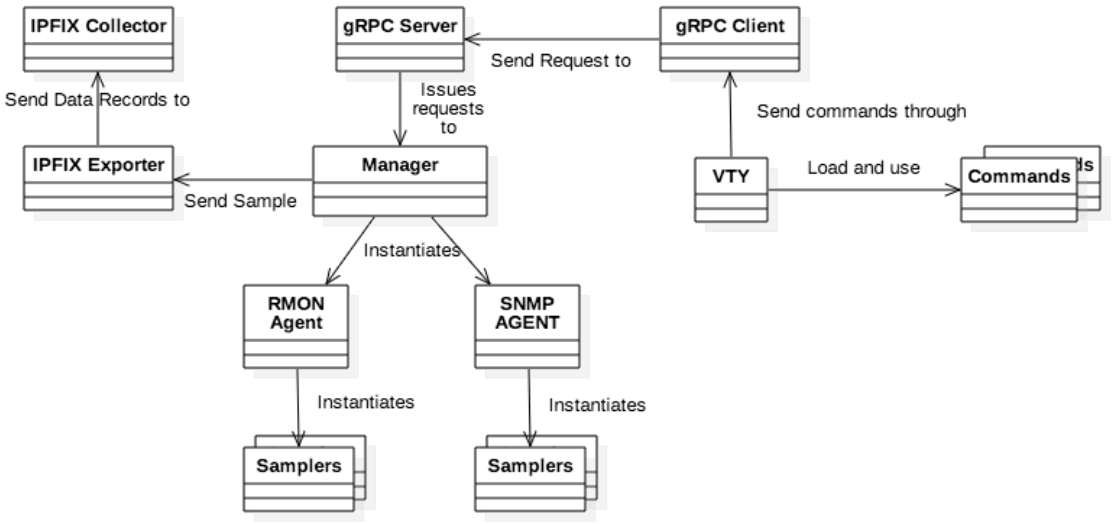


Figura 3-2 – Diseño Node Controller

3.3.2 Operaciones

Para desarrollar el sistema del *NodeController*, se ha hecho uso de la librería PySNMP versión 4.3.7 [20]; puesto que implementa las distintas operaciones de SNMP y sólo hemos tenido que adaptarlas a nuestras necesidades.

De las operaciones disponibles se ha implementado Getnext y Getbulk. Se empezó con Getnext que, como ya se explico en la sección 2.3.1.2, envía una serie de preguntas y obtiene una serie de respuesta. Al principio se decidió implementar una función genérica a

la que se le pasa por argumentos la dirección y puerto donde preguntar, la MIB y los objetos que se deberán solicitar.

Tabla 3-1 – Ejemplo de la operación GetNext

```
1   for (errorIndication,
2           errorStatus,
3           errorIndex,
4           varBinds) in nextCmd(SnmpEngine(),
5                                CommunityData('public', mpModel=1),
6                                UdpTransportTarget((IP,PORT)),
7                                ContextData(),
8                                ObjectType(ObjectIdentity(MIB,Object)),
9                                lexicographicMode=False):
```

Los parámetros a destacar son *mpModel* en *CommunityData* (línea 5 Tabla 3-1), si vale 0 se utiliza SNMPv1 y si es 1 se utiliza SNMPv2c. En *UdpTransportTarget* se le pasa la dirección IP y el Puerto a donde se realizará la petición. Por último, *ObjectType* contiene los distintos *OIDs* de los elementos que se quiere preguntar, dado que trabajar con *OIDs* es tedioso y complicado, hemos hecho uso de *ObjectIdentity* que pasándole la MIB y los nombres de los objetos, este se encarga de convertirlo a *OIDs* (línea 8).

Recordar que para que funcione de manera correcta las operaciones SNMP, en PySNMP deben estar registradas las *MIBs* que se quieren utilizar, en caso contrario dará error; el procedimiento a seguir para poder registrar distintas *MIBs* en el sistema se detalla en el anexo A.B.

La función Getnext la hemos mejorado añadiéndole una parte que se encarga de procesar la respuesta y guardarla en forma de diccionario donde la primera clave es el número de fila de la entrada, de la tabla, y la segunda clave es el nombre de la variable; tal como se muestra en la Tabla 3-2 .

Tabla 3-2 – Procesamiento respuesta Getnext

```
1    for varBind in varBinds:
2                    tipo =varBind[0].prettyPrint()
3                    colnum=re.split('::',tipo)
4                    columnas= colnum[1].split('.',2)
5                    try:
6                        valor = eval(varBind[1].prettyPrint())
7                    except:
8                        valor = varBind[1].prettyPrint()
9                    try:
10                       key1 = eval(columnas[0])
11                   except:
12                       key1 = columnas[0]
13                   try:
14                       key2 = eval(columnas[1])
15                   except:
16                       key2 = columnas[1]
```

```
17                    dict1[key1][key2]=valor
```

Esta función a pesar de estar implementada no se hace uso de ella en el diseño final, pues tal como se dijo en el estado del arte, GetNext realiza una petición por cada campo que se quiere preguntar por lo que en el caso de preguntar la tabla de statistics el número de peticiones eran excesivas, además de que va en contra de nuestro objetivo de monitorizar la red, no saturarla a preguntas. Como solución a GetNext encontramos GetBulk que realiza una sola petición y una sola respuesta, por lo que la carga tanto de los dispositivos como de la red e incluso de nuestro sistema se reduce considerablemente. Comentar que Getbulk pertenece a SNMPv2c por lo que no se podrá utilizar mpModel a 0.

La estructura y el funcionamiento es exactamente la misma y hemos realizado la misma mejora, es decir, añadiendo la parte que se encarga de procesar la respuesta y guardarla en forma de diccionario donde la primera clave es el número de fila de la entrada y la segunda clave es el nombre de la variable. Cabe mencionar que en este caso hemos implementado un Getbulk por cada información a preguntar con el objetivo de no aumentar la complejidad de los Agentes.

La función Set se intentó implementar, para usarla con el grupo *history* de RMON, pero debido a problemas con el conmutador utilizado, finalmente no se desarrolló.

3.3.3 Agentes

En esta sub-sección hablaremos de los agentes implementados y de los *samplers* utilizados para realizar la recolección de los distintos datos.

Los agentes implementados, o *Monitoring Agents,* son el agente RMON y el agente SNMP, se hablará de como se ha implementado, de manera general, debido a que los dos agentes son muy similares. Su diferencia radica en los recolectores de datos que utilizan. El agente RMON será el encargado de configurar los distintos puntos de observación necesarios para los grupos RMON, mientras que el agente SNMP se encargará de los objetos de las MIBs del conmutador Dlink utilizado para las pruebas de este proyecto.

Los dos agentes heredan una interfaz de programación llamada `_Agent` que debe implementar todo *Monitoring Agent* que se integre en el sistema. Las funciones que se deberán implementar, como mínimo son las detalladas en la Tabla 3-3; al heredar los distintos agentes de esa interfaz, nos permite acceder a las variables `self._devideIP` y `self._devicePort` que son la dirección y el puerto donde se mandarán las peticiones.

Tabla 3-3 – Funciones de la interfaz _Agent

```
1    def getDeviceIP(self):
2    def getDevicePort(self):
3    def configure(self, config):
4    def start(self):
5    def disableop(self, observationDomainId,observationPointId):
6    def enableop(self, observationDomainId,observationPointId):
7    def stop(self):
8    def deleteop(self, observationDomainId, observationPointId):
9    def configureObservationPoint(self, observationDomainId,
                                   observationPointId, componentId=None,
                                   componentType=None,
```

```
                                templateId=None,
                                monitoringPeriod=None):
10   def getConfiguredObservationPoints(self, observationDomainId,
                                        observationPointId):
```

Las funciones que se deben implementar para el correcto funcionamiento de un *Monitoring Agent* se detallan seguidamente.

En el método constructor de la interfaz `Agent`, inicializamos las distintas variables que usaremos y también nos permite tener acceso a las instancias de las clases `self.ipfixExporter` y `self.templatesCatalog`.

La función `configure` se encarga de configurar los distintos parámetros de acuerdo a un diccionario que contiene la configuración; esta función llama a `checkConfiguration` que se encarga de comprobar que los parámetros son correctos.

La función `start` inicia el módulo, mientras que la función `stop` se encarga de parar el agente, para ello llama para cada punto de observación a `disableop` para que este pare todos los *samplers*.

La función `enableop` se encarga de arrancar/activar los distintos *samplers* asociados a ese punto de observación; por contra, la función `disableop` desactiva los *samplers*.

La función `deleteop` se encarga de borrar el punto de observación para ello borra la clave, que es una *tupla* del dominio de observación y del punto de observación, del diccionario `self._OP`, que es el que contiene todos los puntos de observación y estos a su vez contienen todos los parámetros y *samplers* asociados con dicho punto de observación.

La función `ConfigureObservationPoint` es la función que se encarga de configurar los distintos puntos de observación que le pasa el *Manager*. Esta función recibe cómo parámetros `observation_domain_id` es el identificador del dominio de observación al que pertenecerá el punto de observación, `observation_point_id` es el identificador del punto de observación, `component_id` es el identificador del componente a monitorizar, `component_type` es el tipo de componente e indica el nombre del *Sampler* a utilizar, `template_id` es el identificador de la plantilla IPFIX que debe de coincidir con la que generará el *Sampler* y, por último, `monitoring_period` es el periodo de monitorización del *Sampler*; esto es la periodicidad con la que se pedirán muestras al dispositivo y se exportarán.

Primero la función `ConfigureObservationPoint` comprueba si la tupla (`observationDomainId`, `observationPointId`) existe o no. Si no existe, asocia los distintos parámetros con dicha tupla, actualiza el exporter con la `templateid` y el `observationDomainId` y se llama a `SamplerFactory` que será el encargado de crear el *sampler* necesario según el `componentType`, por último, configuramos el *sampler* creado y lo activamos.

En el caso de que el punto de observación ya esté creado, comprueba que el `componentType` no ha cambiado para el *sampler*, sino genera una excepción, y actualiza los restantes parámetros; siempre que se hayan pasado como parámetro, actualiza el *exporter*, re-configura el *sampler* y lo vuelve a activar con los datos actualizados.

Por último, tenemos `getConfiguredObservationPoints` que devuelve todos los puntos de observación configurados con su información asociada, es decir, el contenido del

diccionario `self._OP` y, si a esa función le pasamos una tupla (`observationDomainId`, `observationPointId`) devolverá solo los datos asociados a esta.

3.3.4 Samplers

Para la instanciación de *Samplers*, se ha empleado un patrón de diseño software llamado *Factoría*. Este patrón consiste en diseñar una clase que ofrece una función tal que pasando como parámetro un tipo de objeto, nos devuelve una instancia creada de ese objeto. Debido a que todos los *Samplers* siguen una interfaz de programación común, podemos valernos de este patrón de diseño para simplificar la instanciación de los mismos y que el resto de programa que gestiona los *Samplers* sea agnóstico a qué tipo de *Sampler* se trata.

En particular, se ha definido la clase `SamplerFactory` que es la encargada de devolver el *Sampler* instanciado correctamente según el parámetro `componentType` que se indique. Hemos realizado esta estructura debido a que permite aportar extensibilidad al sistema de manera muy sencilla; simplemente creamos una nueva clase *Sampler* que herede de la interfaz `_Sampler`, lo importamos y lo añadimos en el diccionario de *Samplers* llamado CLASS_`MAPPING`, asociándolo con el valor que deberá tener `componentType` para llamar a ese tipo de *Sampler* tal como se explica en el anexo A.C por medio de un ejemplo.

Cada *Sampler* pregunta por una información en concreto, es decir, pregunta por un conjunto de elementos. La estructura de todos los *Samplers* es la misma: periódicamente llaman al método `GetBulk` correspondiente, procesan la respuesta y crean una muestra que deberá ser compatible con aquella definida por la plantilla IPFIX que tenga configurada. Finalmente, añade la información recolectada y procesada a la muestra y la envía al *Manager*.

Por concluir esta sub-sección, indicar que la lista de objetos y MIBs soportadas por los *Samplers* desarrollados en este proyecto se han detallado en del anexo A.E.

3.4 Terminal VTY para pruebas

Para la realización de pruebas se han implementado 3 módulos, el primero es la terminal, y está implementada en el fichero `Main.py` ; en este fichero, se procesan los argumentos de arranque de la terminal así como se añade a dicha terminal los comandos que se han implementado. Dentro de esta carpeta tenemos `Client.py` que contiene las distintas funciones que usan los comandos implementados. Los ficheros que implementan los comandos están en la carpeta `Commands`.

La funciones en `Client.py` se encargan de llamar y pasar los parámetros a las funciones implementadas en el fichero `NCServicerImpl.py`, situado en el lado del *NodeController*; la comunicación entre las funciones de `Client.py` y `NCServicerImpl.py` se realizan utilizando el protocolo gRPC. `NCServicerImpl.py` implementa las funciones que se encargarán de llamar y pasar los argumentos a las funciones pertinentes del *Manager*.

3.5 Conclusiones

En este capítulo, se ha planteado en primer lugar el escenario de aplicación y se ha seleccionado un dispositivo para ser integrado en el sistema de monitorización. En particular, se ha seleccionado el conmutador Dlink DGS-3120-24PC como dispositivo de red a monitorizar.

Seguidamente, se ha presentado la arquitectura general del sistema con sus módulos generales. Destacar en este apartado la generalidad del diseño del sistema que trata de ser lo más extensible posible. Después, se ha descrito la especialización de la arquitectura del sistema que se ha realizado para el dispositivo seleccionado y se ha entrado en detalle en los distintos módulos que la componen.

Finalmente, se ha explicado la terminal de emulación de sistemas de gestión de red desarrollada para realizar las pruebas del sistema desarrollado.

4 Integración, pruebas y resultados

En esta sección describimos las pruebas realizadas sobre el sistema *NodeController*. Para probar el sistema, se ha utilizado el emulador de terminal NCTerm que se ha descrito en la sección A.D y que permite simular los comandos que se podrían recibir de un sistema de gestión de red empleando un protocolo fácilmente integrable en plataformas de monitorización como, por ejemplo, CASTOR [21]. Para su integración en CASTOR, el sistema implementado haría las veces de un controlador de nodos de red adaptable a distintos tipos de dispositivos de red.

Cada una de las pruebas se focaliza en probar uno de los posibles comandos que podría mandar el sistema gestor de red detallando: *i*) el objetivo de la prueba, *ii*) las pre-condiciones que se asumen para llevar a cabo dicha prueba, *iii*) el conjunto de comandos enviados al sistema *NodeController* para desencadenar la prueba, *iv*) la respuesta obtenida del sistema y, si corresponde, *v*) una captura de wireshark con los mensajes relevantes relacionados con la prueba.

4.1 Caso de prueba 1: Arranque y conexión

El objetivo de este caso de prueba es verificar que la configuración de *NodeController* y *NCTerm* es la correcta. Para realizar la configuración en el lado del *NodeController* nos situamos en su carpeta respectiva y dentro de ella encontramos una carpeta llamada data que contiene un fichero denominado "config.json", en este fichero podemos configurar el modo del *Logger* (Por ejemplo, en modo depuración), la IP y Puerto donde el *NodeController* "escucha" para recibir los distintos comandos. Por último, la IP y Puerto donde el *NodeController* enviará las distintas *Samples* de IPFIX. En el lado terminal, la configuración se reduce a modificar el comando "start.sh", situado en la carpeta terminal, es decir, cambiar el argumento que tiene Main.py.

Para arrancar el sistema *NodeController* y la terminal *NCTerm*; nos situamos en las respectivas carpetas de cada uno de ellos y ejecutamos el comando `./start` tal como se indica en la Tabla 4-1 para el caso de *NCTerm*, que además obtendrá el número de versión del manager que corre en *NodeController*. Si *NCTerm* reporta un número de versión válido, por ejemplo 0.1.0, al arrancar, entonces ambos sistemas están correctamente conectados. Además, deberemos ver el cursor "`NCTerm>`", que nos invita a introducir un comando.

Tabla 4-1 – Arranque de *NCTerm* y conexión a *NodeController*

```
$ ./start.sh
Connecting to 0.0.0.0:8000...
manager Version: 0.1.0

NCTerm>
```

Todos los casos de prueba a partir de este punto asumen, como mínimo, que *NodeController* y *NCTerm* están correctamente configurados, en ejecución y que *NCTerm* se conecta correctamente al puerto en el que escucha *NodeController*.

4.2 Caso de prueba 2: Comando "help"

El objetivo de este caso de prueba es probar que funciona correctamente el comando "help". Para poder ejecutar este caso de prueba, no se requiere ninguna pre-condición adicional. Para conocer los distintos comandos que soportan la terminal escribimos el comando "help".

Tabla 4-2 – Comando "help"

```
NCTerm> help
exit     Exit the VTY.
help     Retrieve help for a specific command. Try: help help
agent    Manage the agents.
manager  Manage the Manager.
vty      Manage the VTY.
```

Para listar los sub-comandos que soporta la terminal relativos a la gestión del manager, de los agentes y de los puntos de observación, por ejemplo, escribimos, respectivamente, "help manager", "help agent" y "help agent observationpoint" tal como se indica en la Tabla 4-3.

Tabla 4-3 – Sub-comandos de "help"

```
NCTerm> help manager
start      Start the Manager.
terminate  Terminate the manager.
version    Get manager's version.

NCTerm> help agent
create            Create agent rmon or snmp
delete            Delete agent rmon or snmp
list              List of all agents
modify            Modify agent rmon or snmp
observationpoint  Manage the ops
start             Start all the agents
stop              Stop all the agents


NCTerm> help agent observationpoint
create   Create op(observation point)
delete   Delete op
disable  Disable sampler in op
enable   Enable sampler in op
list     Show all de op configured
modify   Modify op(observation point)
```

4.3 Caso de prueba 3: Comando "agent create"

El objetivo de este caso de prueba es verificar la correcta creación de un agente de monitorización en el sistema. Este caso de prueba asume que, el agente a crear no existe ya en el sistema. Para crear un agente escribimos el comando indicado en la Tabla 4-4. En la Tabla 4-5, se detalla el comando para crear una instancia de agente de tipo "rmon" llamado "rmon1" que conecta con un dispositivo que escucha en la dirección IP 172.16.0.2, puerto 161.

Tabla 4-4 – Comando "agent create"

```
agent create <nombre_agente> <tipo_agente> <ip> <puerto> <mpmodel>
```

Tabla 4-5 – Ejemplo del comando "agent create"

```
NCTerm> agent create rmon1 rmon 172.16.0.2 161 1
Create agents rmon or snmp request send.
```

En el manager, si está en modo de depuración, deberá salir la confirmación de los parámetros de creación del nuevo agente como se detalla en la Tabla 4-6.

Tabla 4-6 – Respuesta del *NodeController* al comando "agent create"

```
[2018-05-14 19:07:02,510] DEBUG   Manager   : Init in RMON
[2018-05-14 19:07:02,511] DEBUG   Manager   : Configuration parameters
RMON Agent
[2018-05-14 19:07:02,511] DEBUG   Manager   : 172.16.0.2
[2018-05-14 19:07:02,511] DEBUG   Manager   : 161
[2018-05-14 19:07:02,511] DEBUG   Manager   : 1
[2018-05-14 19:07:02,511] DEBUG   Manager   : Passing parameters to
RMON agent
```

4.4 Caso de prueba 4: Comando "agent list"

En este caso de prueba vamos a mostrar la lista de agentes creados en el *NodeController*. Este caso de prueba no tiene ninguna pre-condición adicional. En caso de no haber creado algún agente, la lista saldrá vacía. El comando a ejecutar así como el resultado del mismo, se muestran en la Tabla 4-7.

Tabla 4-7 – Ejemplo del comando "agent list"

```
NCTerm> agent list
Agents:
  NameAgent: rmon1
  Type: Agent_RMON
  IP: 172.16.0.2
  Port: 161
  mpModel: 1
```

4.5 Caso de prueba 5: Comando "agent modify"

En siguiente caso de prueba modificaremos los parámetros de un agente existente. Para poder ejecutar este comando debe de existir el agente. Para modificar un agente, ejecutaremos el comando indicado en la Tabla 4-8.

Tabla 4-8 – Comando "agent modify"

```
agent modify <nombre_agente> <tipo_agente> <ip> <puerto> <mpmodel>
```

En la Tabla 4-9, se muestra un ejemplo de este comando donde se modifica el agente "rmon1" para que apunte al agente de dispositivo con dirección IP 10.1.1.1 y puerto 150.

Tabla 4-9 – Ejemplo del comando "agent modify"

```
NCTerm> agent modify rmon1 rmon 10.1.1.1 150 1
Correct
```

Para comprobar el funcionamiento correcto del comando recurrimos al comando "agent list"; tal como se ve en la Tabla 4-10, los parámetros han sido modificados.

Tabla 4-10 – Lista de agentes después de ejecutar el comando "agent modify"

```
NCTerm> agent list
Agents:
  NameAgent: rmon1
  Type: Agent_RMON
  IP: 10.1.1.1
  Port: 150
  mpModel: 1
 ------------------------
```

En caso de intentar modificar un agente que no ha sido creado con anterioridad, obtendremos el error detallado en la Tabla 4-11.

Tabla 4-11 – Error en "agent modify"

```
NCTerm> agent modify rmon1 rmon 10.1.1.1 161 1
Failed
```

4.6 Caso de prueba 6: Comando "agent start"

En este caso de prueba, arrancaremos los agentes configurados en NodeController. Para poder ejecutar este comando debe de haberse creado previamente un agente. Para arrancar todos los agentes, escribimos el comando indicado en la Tabla 4-12.

Tabla 4-12 – Comando "agent start"

```
agent start
```

En el manager, si está en modo de depuración, deberá salir la confirmación del arranque del agente como se detalla en la Tabla 4-13.

Tabla 4-13 – Respuesta del *NodeController* al comando "agent start"

```
[2018-05-14 19:07:33,871] DEBUG   Manager   : startagents in Manager
[2018-05-14 19:07:33,871] DEBUG   Manager   : start in RMON Agent
```

4.7 Caso de prueba 7: Comando "agent observationpoint create"

En este caso de prueba, crearemos un punto de observación que a su vez creará un *Sampler*. Para poder ejecutar este comando previamente se ha debido de crear un agente. Para crear un punto de observación, debemos de escribir el comando indicado en la Tabla 4-14.

Tabla 4-14 – Comando "agent observationpoint create"

```
agent observationpoint create <nombre_Agente> <observation_domain_id>
<observation_point_id> <component_id> <component_type> <template_id>
<monitoring_period>
```

Este comando requiere los siguientes parámetros: "nombre_agente" es el nombre del agente creado anteriormente, "observation_domain_id" es el identificador del dominio de observación al que pertenecerá el punto de observación, "observation_point_id" es el identificador del punto de observación, "component_id" es el identificador del componente a monitorizar, "component_type" es el tipo de componente e indica el nombre del *Sampler* a utilizar, "template_id" es el identificador de la plantilla IPFIX que debe de coincidir con la que generará el *Sampler* y se haya registrado en el fichero de "IPFIX_templates.json" y, por último, "monitoring_period" es el periodo de monitorización del *Sampler*; esto es la periodicidad con la que se pedirán muestras al dispositivo y se exportarán.

Para crear un punto de observación perteneciente al agente "rmon1", con identificador de dominio de observación "1", identificador de punto de observación "2", identificador de componente "4" y tipo de componente "statistics", plantilla IPFIX "501" y periodo de monitorización de "20" segundos, escribimos el comando indicado en la Tabla 4-15.

Tabla 4-15 – Ejemplo "agent observationpoint create"

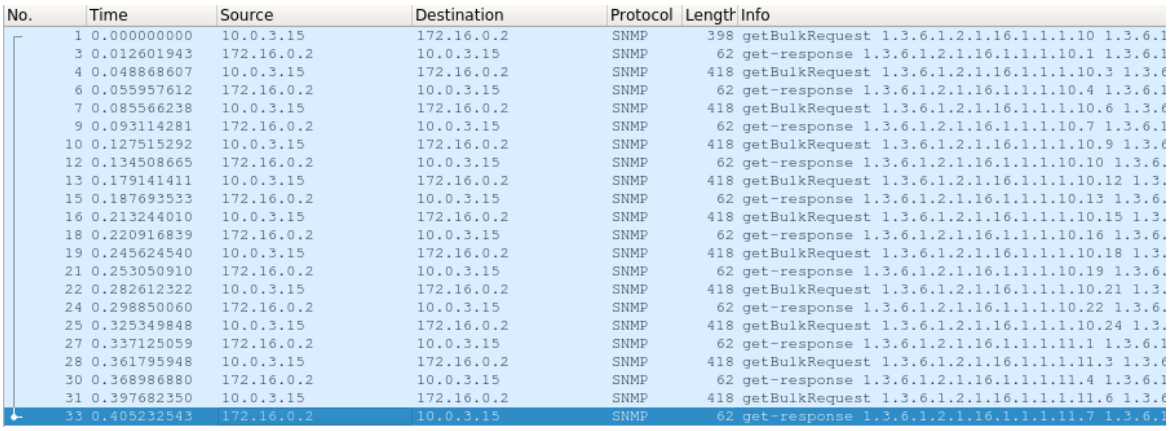
```
NCTerm> agent observationpoint create rmon1 1 2 4 statistics 501 20
Create agents rmon or snmp request send.
```

En el manager, si está en modo de depuración, deberá salir la confirmación de la creación del *Sampler* como se detalla en la Tabla 4-16.

Tabla 4-16 – Respuesta del *NodeController* al comando "`agent observationpoint start`"

```
[2018-05-14 19:18:34,912] DEBUG   Manager   : Creating Sampler in RMON
[2018-05-14 19:18:34,912] DEBUG   Manager   : Sampler Factory in RMON
Agent
[2018-05-14 19:18:34,912] DEBUG   Manager   : Estamos en sampler
factory
[2018-05-14 19:18:34,913] DEBUG   Manager   : retornamos sample desde
sampler factory
[2018-05-14 19:18:34,913] DEBUG   Manager   : init in statistics
sampler
[2018-05-14 19:18:34,913] DEBUG   Manager   : configurating sampler in
RMON Agent
[2018-05-14 19:18:34,913] DEBUG   Manager   : estamos en configuracion
del sampler
[2018-05-14 19:18:34,913] DEBUG   Manager   : check in the Statistics
Sampler if the templateid matches the one that has been passed
[2018-05-14 19:18:34,914] DEBUG   Manager   : Starting sampler in RMON
[2018-05-14 19:18:34,914] DEBUG   Manager   : Estamos en start del
sampler
[2018-05-14 19:18:34,914] DEBUG   Manager   : In  _scheduleNext
[2018-05-14 19:18:44,915] DEBUG   Manager   : do_sampling in
statistics Sampler
[2018-05-14 19:18:44,915] DEBUG   Manager   : getbulkstatistics
[2018-05-14 19:18:45,434] DEBUG   Manager   : runcallbacks in
statistics sampler
[2018-05-14 19:18:45,434] DEBUG   Manager   : Receveir in Manager
[2018-05-14 19:18:45,438] DEBUG   Manager   : In  _scheduleNext
[2018-05-14 19:18:55,441] DEBUG   Manager   : do_sampling in
statistics Sampler
[2018-05-14 19:18:55,442] DEBUG   Manager   : getbulkstatistics
[2018-05-14 19:18:55,890] DEBUG   Manager   : runcallbacks in
statistics sampler
[2018-05-14 19:18:55,890] DEBUG   Manager   : Receveir in Manager
[2018-05-14 19:18:55,893] DEBUG   Manager   : In  _scheduleNext
```

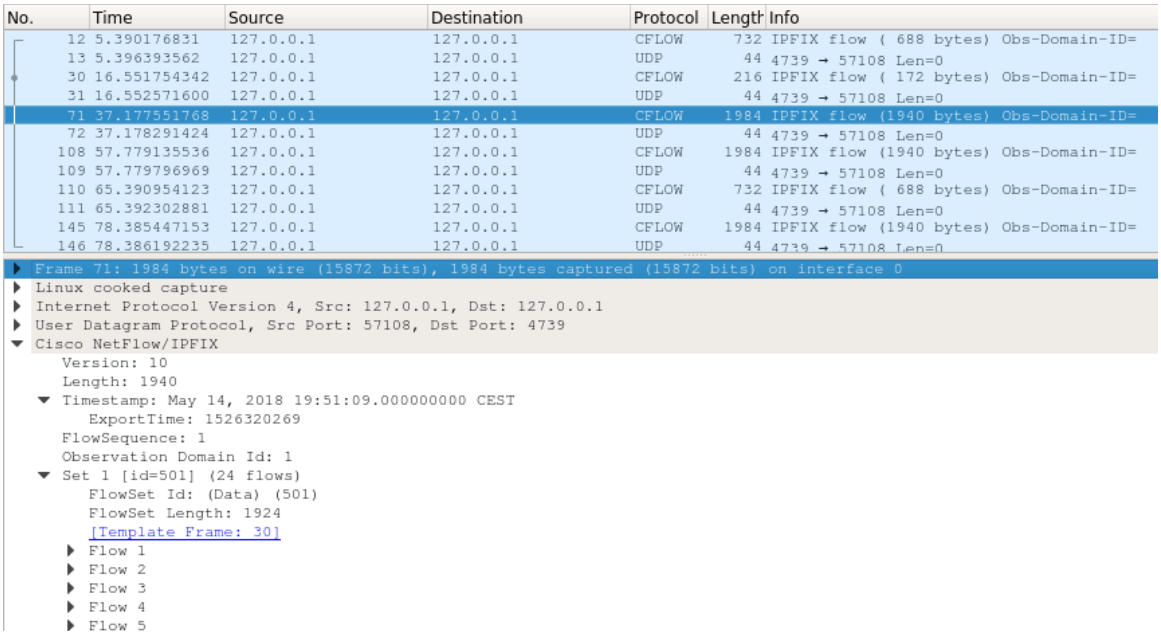
Tras ejecutar satisfactoriamente el comando, si activamos una captura de tráfico de red en la interfaz del dispositivo o del *NodeController*, por ejemplo usando Wireshark como se ilustra en la figura Figura 4-1, deberíamos observar que periódicamente, el *Sampler* preguntará al dispositivo por la tabla de estadísticas (`statistics`) de RMON mediante la operación "*GetBulk*" y recibirá dicha tabla del dispositivo.



No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	10.0.3.15	172.16.0.2	SNMP	398	getBulkRequest 1.3.6.1.2.1.16.1.1.1.10 1.3.6.1
3	0.012601943	172.16.0.2	10.0.3.15	SNMP	62	get-response 1.3.6.1.2.1.16.1.1.1.10.1 1.3.6.1
4	0.048868607	10.0.3.15	172.16.0.2	SNMP	418	getBulkRequest 1.3.6.1.2.1.16.1.1.1.10.3 1.3.6
6	0.055957612	172.16.0.2	10.0.3.15	SNMP	62	get-response 1.3.6.1.2.1.16.1.1.1.10.4 1.3.6.1
7	0.085566238	10.0.3.15	172.16.0.2	SNMP	418	getBulkRequest 1.3.6.1.2.1.16.1.1.1.10.6 1.3.6
9	0.093114281	172.16.0.2	10.0.3.15	SNMP	62	get-response 1.3.6.1.2.1.16.1.1.1.10.7 1.3.6.1
10	0.127515292	10.0.3.15	172.16.0.2	SNMP	418	getBulkRequest 1.3.6.1.2.1.16.1.1.1.10.9 1.3.6
12	0.134508665	172.16.0.2	10.0.3.15	SNMP	62	get-response 1.3.6.1.2.1.16.1.1.1.10.10 1.3.6.
13	0.179141411	10.0.3.15	172.16.0.2	SNMP	418	getBulkRequest 1.3.6.1.2.1.16.1.1.1.10.12 1.3.
15	0.187693533	172.16.0.2	10.0.3.15	SNMP	62	get-response 1.3.6.1.2.1.16.1.1.1.10.13 1.3.6.
16	0.213244010	10.0.3.15	172.16.0.2	SNMP	418	getBulkRequest 1.3.6.1.2.1.16.1.1.1.10.15 1.3.
18	0.220916839	172.16.0.2	10.0.3.15	SNMP	62	get-response 1.3.6.1.2.1.16.1.1.1.10.16 1.3.6.
19	0.245624540	10.0.3.15	172.16.0.2	SNMP	418	getBulkRequest 1.3.6.1.2.1.16.1.1.1.10.18 1.3.
21	0.253050910	172.16.0.2	10.0.3.15	SNMP	62	get-response 1.3.6.1.2.1.16.1.1.1.10.19 1.3.6.
22	0.282612322	10.0.3.15	172.16.0.2	SNMP	418	getBulkRequest 1.3.6.1.2.1.16.1.1.1.10.21 1.3.
24	0.298850060	172.16.0.2	10.0.3.15	SNMP	62	get-response 1.3.6.1.2.1.16.1.1.1.10.22 1.3.6.
25	0.325349848	10.0.3.15	172.16.0.2	SNMP	418	getBulkRequest 1.3.6.1.2.1.16.1.1.1.10.24 1.3.
27	0.337125059	172.16.0.2	10.0.3.15	SNMP	62	get-response 1.3.6.1.2.1.16.1.1.1.11.1 1.3.6.1
28	0.361795948	10.0.3.15	172.16.0.2	SNMP	418	getBulkRequest 1.3.6.1.2.1.16.1.1.1.11.3 1.3.6
30	0.368986880	172.16.0.2	10.0.3.15	SNMP	62	get-response 1.3.6.1.2.1.16.1.1.1.11.4 1.3.6.1
31	0.397682350	10.0.3.15	172.16.0.2	SNMP	418	getBulkRequest 1.3.6.1.2.1.16.1.1.1.11.6 1.3.6
33	0.405232543	172.16.0.2	10.0.3.15	SNMP	62	get-response 1.3.6.1.2.1.16.1.1.1.11.7 1.3.6.1

Figura 4-1 – Captura de muestras de la tabla "`statistics`"

Así mismo, la captura del Wireshark de IPFIX con las muestras que se exportan al sistema recolector que esté configurado se pueden ver en la Figura 4-2.



No.	Time	Source	Destination	Protocol	Length	Info
12	5.390176831	127.0.0.1	127.0.0.1	CFLOW	732	IPFIX flow (688 bytes) Obs-Domain-ID=
13	5.396393562	127.0.0.1	127.0.0.1	UDP	44	4739 → 57108 Len=0
30	16.551754342	127.0.0.1	127.0.0.1	CFLOW	216	IPFIX flow (172 bytes) Obs-Domain-ID=
31	16.552571600	127.0.0.1	127.0.0.1	UDP	44	4739 → 57108 Len=0
71	37.177551768	127.0.0.1	127.0.0.1	CFLOW	1984	IPFIX flow (1940 bytes) Obs-Domain-ID=
72	37.178291424	127.0.0.1	127.0.0.1	UDP	44	4739 → 57108 Len=0
108	57.779135536	127.0.0.1	127.0.0.1	CFLOW	1984	IPFIX flow (1940 bytes) Obs-Domain-ID=
109	57.779796969	127.0.0.1	127.0.0.1	UDP	44	4739 → 57108 Len=0
110	65.390954123	127.0.0.1	127.0.0.1	CFLOW	732	IPFIX flow (688 bytes) Obs-Domain-ID=
111	65.392302881	127.0.0.1	127.0.0.1	UDP	44	4739 → 57108 Len=0
145	78.385447153	127.0.0.1	127.0.0.1	CFLOW	1984	IPFIX flow (1940 bytes) Obs-Domain-ID=
146	78.386192235	127.0.0.1	127.0.0.1	UDP	44	4739 → 57108 Len=0

```
▶ Frame 71: 1984 bytes on wire (15872 bits), 1984 bytes captured (15872 bits) on interface 0
▶ Linux cooked capture
▶ Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
▶ User Datagram Protocol, Src Port: 57108, Dst Port: 4739
▼ Cisco NetFlow/IPFIX
    Version: 10
    Length: 1940
  ▼ Timestamp: May 14, 2018 19:51:09.000000000 CEST
      ExportTime: 1526320269
    FlowSequence: 1
    Observation Domain Id: 1
  ▼ Set 1 [id=501] (24 flows)
      FlowSet Id: (Data) (501)
      FlowSet Length: 1924
      [Template Frame: 30]
    ▶ Flow 1
    ▶ Flow 2
    ▶ Flow 3
    ▶ Flow 4
    ▶ Flow 5
```

Figura 4-2 – Muestras exportadas mediante IPFIX al sistema recolector

4.8 Caso de prueba 8: Comando "`agent observationpoint list`"

En este caso de prueba, se verificará el correcto funcionamiento del comando "`agent observationpoint list`", que es el encargado de mostrarnos todos los puntos de observación configurados. Para poder ejecutar este comando, es conveniente haber configurado algún punto de observación; en caso contrario, se mostrará una lista vacía. El comando a ejecutar así como el resultado del mismo, se muestran en la Tabla 4-17.

Tabla 4-17 – Ejemplo "agent observationpoint list"

```
NCTerm> agent observationpoint list
Observation Points:

  ObservationDomain: 1
  ObservationPoint: 2
    Enable: False
    Agent: rmon1
    ComponentId: 4
    ComponentType: statistics
    templateId: 501
    monitoringPeriod: 20
------------------------
```

4.9 Caso de prueba 9: Comando "agent observationpoint modify"

Con este caso de prueba modificaremos un punto de observación existente. Debemos de tener en cuenta que existen parámetros que no pueden ser modificados; estos parámetros son: el identificador del punto de observación, el identificador del dominio de observación y el tipo de componente. Como pre-requisito, debemos de tener configurado el punto de observación. La sintaxis del comando, detallada en la Tabla 4-18, es igual que la del comando "`agent observationpoint create`", pero en este caso se debe de tener en cuenta las consideraciones anteriores.

Tabla 4-18 – Comando "agent observationpoint modify"

```
agent observationpoint modify <nombre_Agente> <observation_domain_id>
<observation_point_id> <component_id>  <component_type> <template_id>
<monitoring_period>
```

Para modificar un punto de observación perteneciente al agente "`rmon1`", con identificador de dominio de observación "`1`", identificador de punto de observación "`2`", identificador de componente "`10`" y tipo de componente "`statistics`", plantilla IPFIX "`501`" y pasar su periodo de monitorización a "`10`" segundos, escribimos el comando indicado en la Tabla 4-19.

Tabla 4-19 – Ejemplo "agent observationpoint modify"

```
agent observationpoint modify rmon1 1 2 10 statistics 501 10
```

Para comprobar que se han realizado los cambios pertinentes, recurrimos al comando "`agent observationpoint list`", el cual nos muestra el punto de observación actualizado, como se puede ver en la Tabla 4-20.

Tabla 4-20 – Lista de puntos de observacion modificados

```
NCTerm> agent observationpoint list
Observation Points:

  ObservationDomain: 1
  ObservationPoint: 2
    Enable: False
    Agent: rmon1
    ComponentId: 10
    ComponentType: statistics
    templateId: 501
    monitoringPeriod: 10
------------------------
```

Por el contrario, si intentamos modificar un punto de observación que no ha sido creado con anterioridad, nos saldrá un error como en el ejemplo de la Tabla 4-21.

Tabla 4-21 – Error "agent observationpoint modify"

```
NCTerm> agent observationpoint modify rmon1 4 2 12 statistis 501 32
Fail modify observation point
```

4.10 Caso de prueba 10: Comando "agent observationpoint disable"

En este caso de prueba, probaremos a desactivar un punto de observación existente. Para ello debemos de tener previamente un punto de observación configurado. Para poder desactivarlo, ejecutaremos el comando indicado en la Tabla 4-22. Los parámetros a indicar son el nombre del agente responsable del punto de observación y los identificadores del dominio y del punto de observación.

Tabla 4-22 – Comando "agent observationpoint disable"

```
agent observation point <nombre_agente>  <observation_domain>
<observation_point>
```

Para desactivar un punto de observación perteneciente al agente "rmon1", con identificador de dominio de observación "1" e identificador de punto de observación "2", escribimos el comando indicado en la Tabla 4-23.

Tabla 4-23 – Ejemplo "agent observationpoint disable"

```
agent observationpoint disable rmon1 1 2
```

En la Tabla 4-24, podemos ver con el comando "`agent observationpoint list`" como antes estaba activado y ahora desactivado, en el Manager, si está en modo depuración, se vería como el temporizador del *Sampler* se ha parado.

Tabla 4-24 – Lista de puntos de observación antes y después de desactivarlo

```
NCTerm> agent observation list
Observation Points:

  ObservationDomain: 1
  ObservationPoint: 2
    Enable: True
    Agent: rmon1
    ComponentId: 4
    ComponentType: statistics
    templateId: 501
    monitoringPeriod: 20
------------------------
```

```
NCTerm> agent observationpoint list
Observation Points:

  ObservationDomain: 1
  ObservationPoint: 2
    Enable: False
    Agent: rmon1
    ComponentId: 4
    ComponentType: statistics
    templateId: 501
    monitoringPeriod: 20
------------------------
```

Por el contrario, si intentamos desactivar un punto de observación no creado anteriormente, nos saldrá el mensaje de error indicado en la Tabla 4-25.

Tabla 4-25 – Error "`agent observationpoint disable`"

```
NCTerm> agent observationpoint disable rmon1 2 4
Fail
```

4.11 Caso de prueba 11: Comando "`agent observationpoint enable`"

Ahora, en este caso de prueba, vamos a probar a activar el punto de observación de nuevo. Como prerrequisito para ejecutar este comando, debemos de tener creado el punto de observación y debe estar desactivado. La sintaxis del comando a ejecutar se detalla en la Tabla 4-26.

Tabla 4-26 – Comando "agent observationpoint enable"

```
agent observationpoint enable <nombre_agente> <observation_domain>
<observation_point>
```

Para activar un punto de observación perteneciente al agente "rmon1", con identificador de dominio de observación "1" e identificador de punto de observación "2", escribimos el comando indicado en la Tabla 4-27.

Tabla 4-27 – Ejemplo "agent observationpoint enable"

```
NCTerm> agent observationpoint enable rmon1 1 2
All correct
```

Y comprobamos que el comando ha funcionamiento correctamente utilizando el comando "agent observationpoint list" que nos muestra que el punto de observación está activado, como se puede ver en la Tabla 4-28.

Tabla 4-28 – Lista de puntos de observación después de activarlos

```
NCTerm> agent observationpoint list
Observation Points:

  ObservationDomain: 1
  ObservationPoint: 2
    Enable: True
    Agent: rmon1
    ComponentId: 4
    ComponentType: statistics
    templateId: 501
    monitoringPeriod: 20
------------------------
```

Si intentamos activar un punto de observación no creado anteriormente nos saldrá el mensaje de error detallado en la Tabla 4-29.

Tabla 4-29 – Error "agent observationpoint enable"

```
NCTerm> agent observationpoint enable rmon1 2 4
Fail
```

4.12 Caso de prueba 12: Comando "agent observationpoint delete"

En este caso de prueba, se probará a eliminar un punto de observación. Como prerrequisito necesitamos tener creado ese observation point. El comando a ejecutar se detalla en la Tabla 4-30.

Tabla 4-30 – Comando "agent observationpoint delete"

```
agent observationpoint delete <nombre_agente> <observation_domain>
<observation_point>
```

Por ejemplo, si queremos eliminar el punto de observación con identificador de dominio "1" e identificador de punto de observación "2" del agente "rmon1", que hemos utilizado para estos casos de prueba, el comando sería el indicado en la Tabla 4-31.

Tabla 4-31 – Ejemplo "agent observationpoint delete"

```
NCTerm> agent observationpoint delete rmon1 1 2
All Correct
```

Y para comprobar que ha funcionado de manera correcta, recurrimos al comando "agent observationpoint list"; como era de esperar, ese punto de observación habrá sido eliminado, como se ve en la Tabla 4-32.

Tabla 4-32 – Lista de puntos de observación después del borrado

```
NCTerm> agent observationpoint list
Observation Points:
```

Si intentamos borrar un punto de observación inexistente, se nos reportará el mensaje de error detallado en la Tabla 4-33.

Tabla 4-33 – Error "agent observationpoint delete"

```
NCTerm> agent observationpoint delete rmon1 2 4
Fail
```

4.13 Caso de prueba 13: Comando "agent stop"

En este caso de prueba, pararemos los agentes configurados en *NodeController*. Para poder ejecutar este comando debe de haberse creado previamente un agente. Para parar todos los agentes escribimos el comando indicado en la Tabla 4-34:

Tabla 4-34 – Comando "`agent stop`"

```
agent stop
```

En el manager, si está en modo de depuración, deberá salir la confirmación de parada del agente como se detalla en la Tabla 4-35.

Tabla 4-35 – Respuesta del *NodeController* al comando "`agent stop`"

```
[2018-05-14 19:07:33,871] DEBUG   Manager   : stopagents in Manager
[2018-05-14 19:07:33,871] DEBUG   Manager   : stop in RMON Agent
```

4.14 Caso de prueba 14: Comando "`agent delete`"

En este caso de prueba, borraremos el agente que llevamos utilizando en esta explicación, es decir "`rmon1`". Para poder ejecutar este caso de prueba, debemos de tener el agente creado. El comando a utilizar se detalla en la Tabla 4-36.

Tabla 4-36 – Comando "`agent delete`"

```
agent delete <nombre_agente>
```

En la Tabla 4-37, se muestra un ejemplo de este comando donde se borra "`rmon1`".

Tabla 4-37 – Ejemplo "`agent delete`"

```
NCTerm> agent delete rmon1
All correct
```

Para comprobar que hemos borrado el agente recurrimos al comando "`agent list`"; tal como se muestra en la Tabla 4-38, como era de esperar, el agente ha sido borrado, por lo que la lista sale vacía.

Tabla 4-38 – Error "`agent delete`"

```
NCTerm> agent list
Agents:
```

4.15 Caso de prueba 15: Comando "`manager terminate`"

En este último caso de prueba, apagaremos el manager del *NodeController* desde la terminal, provocando que el proceso termine. Como prerrequisito necesitaremos tener encendido el manager. Ejecutaremos el comando indicado en la Tabla 4-39 y deberíamos obtener la respuesta indicada. A los pocos segundos, el *NodeController* debería apagarse.

Tabla 4-39 – Ejemplo "`manager termiante`"

```
NCTerm> manager terminate
Terminate request sent.
```

4.16 Conclusiones

En este capítulo, se ha explicado todas las funcionalidades de nuestro Sistema. Se han realizado 15 casos de prueba con el objetivo de demostrar el correcto funcionamiento del sistema, en cada caso se ha utilizado uno de los comandos que soporta. Podemos concluir que todas las funcionalidades propuestas han sido implementadas en el sistema y verificadas satisfactoriamente.

5 Conclusiones y trabajo futuro

5.1 Conclusiones

El futuro incremento de demanda en ancho de banda y estabilidad motivará que las compañías operadoras de redes de comunicaciones optimicen el uso de sus redes y ofrezcan un servicio de mejor calidad y precio a los usuarios. Para poder respetar los acuerdos de calidad de servicio y disponibilidad es necesario monitorizar la red con el objetivo de conocer que está ocurriendo en la misma y si los cambios que se produzcan pueden afectar a la calidad o disponibilidad de dicho servicio.

Para monitorizar la red, es necesario que esta emplee dispositivos con capacidades de monitorización; de lo contrario, será necesario instalar sondas en la red que nos provean de estos datos. La cantidad de protocolos de monitorización que pueden soportar los dispositivos de red y las sondas pueden ser muy variados por lo que existe una gran heterogeneidad en los relativo a los protocolos de monitorización.

En este proyecto se ha definido como objetivo implementar un sistema que sea capaz de unificar las interfaces de monitorización de distintos dispositivos de red, tratando de recolectar los datos de monitorización que estos reportan y traducirlos a un formato común. Esto permite que un sistema de gestión de red pueda obtener dichos datos a través de una única interfaz reduciendo su complejidad.

En el estado del arte hemos repasado los conceptos importantes relacionados con los componentes principales que conforman una red de computadores, así como las capas en las que se organizan los protocolos de red. También se han tratado los conceptos relativos a la monitorización de redes y distintos protocolos estandarizados que normalmente se emplean a tal efecto.

Luego, en el capítulo “Diseño y desarrollo” se propuesto un dispositivo de red a ser monitorizado por el sistema propuesto, en concreto un conmutador de paquetes Dlink DGS-3120-24PC, y se ha presentado la arquitectura general del sistema. Seguidamente, se ha especializado la mencionada arquitectura implementando el protocolo SNMP con las MIBs que ofrece Dlink para el dispositivo, así como las MIBs estandarizadas de RMON que soportaba dicho dispositivo. El proyecto se ha llevado a cabo de forma que sea fácilmente extensible a otros dispositivos que soporten distintas MIBs y otros protocolos.

Por último, en el capítulo “Integración, pruebas y resultados” se ha explicado todas las funcionalidades de nuestro sistema y se ha demostrado el correcto funcionamiento del mismo, mediante la realización de 15 casos de prueba.

Para la realización de este Trabajo de Fin de Grado se ha requerido una amplia variedad de conocimientos relacionados con las redes de computadores, por lo que los conocimientos adquiridos en las asignaturas de redes pertenecientes al Grado en Ingeniería en Tecnologías y Servicios de Telecomunicación han sido indispensables para la realización del proyecto.

En Arquitectura de Redes I se estudió las distintas capas de la red, en concreto el nivel de aplicación, de transporte y de red; también se aprendió a analizar el tráfico de red y a implementar sistemas que se comunicaran por la red empleando protocolos estandarizados.

En Arquitectura de Redes II, se estudió el funcionamiento de las redes actuales, así como protocolos de comunicación y mecanismos de seguridad y gestión que se pueden implementar en las redes. En particular, se profundizó más en la captura y análisis de tráfico de red.

Redes multimedia destacó la importancia de medir y respectar la calidad de servicio que ofrecen las operadoras y es uno de los pilares sobre el que se sustenta este proyecto: proveer a las operadoras de redes de comunicaciones de sistemas que les faciliten la tarea de ofrecer servicios de calidad y así respectar los acuerdos de servicios con sus clientes. Colateralmente, en Redes Multimedia se profundizó en el uso de Wireshark así como en protocolos estandarizados para la transmisión de contenido multimedia en tiempo real.

Finalmente, cabe mencionar las asignaturas de programación (Programación I y Programación II), que fueron las primeras que nos introdujeron en el mundo de la programación y sentaron las bases para el desarrollo de sistemas software complejos.

Además, el código de este proyecto ha sido liberado, bajo licencia GPL-3.0, en GitHub [25].

5.2 Trabajo futuro

Dado que el diseño del sistema realizado en este proyecto se ha pensado para que sea fácilmente extensible, el trabajo desarrollado en este proyecto podría extenderse de diversas formas:

- Por un lado, se podría ampliar la cantidad de MIBs que soporta el sistema, añadiendo nuevos *Samplers*.
- Por otro lado, se podría extender el sistema con nuevos *Monitoring Agents* que soporten otros protocolos de monitorización.
- Se puede probar con otros dispositivos de red e implementar los módulos que estos requieran.
- Se podría aumentar el conjunto de comandos de control que soporta el sistema, extendiendo nuevas funcionalidades, quizá relacionado con los puntos anteriores.
- Se podría mejorar la implementación del módulo GetBulk con el objetivo de generalizarlo y evitar tener que re-implementar funciones nuevas para cada *Sampler*.
- Finalmente, se podría llevar a cabo la integración final del sistema en una plataforma de monitorización, como puede ser, por ejemplo, la plataforma CASTOR [21].

Por tanto, se pone de manifiesto la viabilidad de definir nuevos TFGs que partan de este proyecto y permitan mejorar el sistema base desarrollado.

Referencias

[1] "Cisco Visual Networking. Index: Forecast and Methodology, 2016-2021", Junio 2017. [on-line] https://www.cisco.com/c/en/us/solutions/service-provider/visual-networking-index-vni/index.html

[2] S. Waldbusser, C.Kalbfleisch, D. Romascanu, "Introduction to the Remote Monitoring (RMON) Family of MIB Modules", IETF RFC 2819, Agosto 2003. [on-line] https://tools.ietf.org/html/rfc2819

[3] James F. Kurose, Keith W. Ross, "Redes de computadoras", Pearson, 2010.

[4] Networking Basics [on-line]. https://www.cisco.com/c/en/us/solutions/small-business/resource-center/connect-employees-offices/networking-basics.html

[5] "Darpa Internet Program Protocol Specification", IETF RFC 791, Septiembre 1981. [on-line] https://tools.ietf.org/html/rfc791

[6] J. Postel, "User Datagram Protocol", IETF RFC 768, Agosto 1980. [on-line] https://tools.ietf.org/html/rfc768

[7] J. Postel, "Transmission Control Protocol", IETF RFC 793, Septiembre 1981. [on-line] https://tools.ietf.org/html/rfc793

[8] Douglas R. Mauro, Kevin J. Schmidt, "Essential SNMP", O'Reilly, Septiembre 2005.

[9] SNMP Design [on-line]. http://pysnmp.sourceforge.net/docs/snmp-design.html

[10] K. McCloghrie, "Management Information Base for Network Management of TCP/IP-based internets", IETF RFC 1156, Mayo 1990. [on-line] https://tools.ietf.org/html/rfc1156

[11] K. McCloghrie, J. Schoenwaelder, J. Case, M. Rose, "Structure of Management Information Version 2 (SMIv2)", IETF RFC 2578, Abril 1999. [on-line] https://tools.ietf.org/html/rfc2578

[12] C. Wallace, C. Gardiner, "ASN.1 Translation", IETF RFC 6025, Octubre 2010. [on-line] https://tools.ietf.org/html/rfc6025

[13] Network Management Software [on-line]. https://www.networkmanagementsoftware.com/snmp-tutorial-part-2-rounding-out-the-basics/

[14] B. Claise, B. Trammell, P. Aitken, "Specification of the IP Flow Information Export (IPFIX) Protocol for the Exchange of Flow Information", IETF RFC 7011, Septiembre 2013. [on-line] https://tools.ietf.org/html/rfc7011

[15] IANA.org – IPFIX Information Entities [on-line]. https://www.iana.org/assignments/ipfix/ipfix.xhtml

[16] gRPC [on-line]. https://grpc.io/

[17] SNMP Agent Simulator [on-line] http://snmplabs.com/snmpsim/index.html

[18] GNS3 – The software that empowers network professionals [on-line] https://www.gns3.com/

[19] Descripción del conmutador Dlink DGS-3120-24PC-SI [on-line] http://www.dlinkla.com/dgs-3120-24pc-si

[20] Librería PySNMP [on-line]. http://pysnmp.sourceforge.net/docs/snmp-design.html

[21] Ll. Gifre, J.-L. Izquierdo-Zaragoza, M. Ruiz, and L. Velasco, "Autonomic Disaggregated Multilayer Networking," in IEEE/OSA Journal of Optical Communications and Networking (JOCN), vol. 10, pp. 482-492, 2018.

[22] Python.org [on-line]. https://www.python.org/downloads/

[23] Pip.pypa.io [on-line]. https://pip.pypa.io/en/stable/

[24] gRPC – Setup and Getting Started [on-line]. https://grpc.io/docs/quickstart/python.html

[25] GitHub [on-line]. https://github.com/lgifre/NodeController

[26] GitHub [on-line]. https://github.com/lgifre/pyIPFIX

[27] GitHub [on-line]. https://github.com/lgifre/pyTools

Anexos

A Manual de instalación

En este anexo, se describirán los pasos necesarios para instalar las dependencias del sistema recolector de datos desarrollados en este proyecto; en particular, necesitamos tener instalado Python, PySNMP y GRPC.

A.1 Instalación del lenguaje de programación Python

Para asegurar que el sistema funciona correctamente, deberemos instalar Python 2.7.14; accedemos a la web de Python [22] y seguiremos los pasos de instalación que se definan para nuestra plataforma. También deberemos instalar el gestor de librerías PIP de Python siguiendo los pasos que se indica en [23].

A.2 Instalación de la librería PySNMP

Dado que las versiones de la librería pysnmp que utilicemos deben ser compatible con la utilizada para este proyecto, escribimos el siguiente comando sin importar si instalamos una versión más reciente a la utilizada durante el desarrollo de este proyecto.

Pip se debe instalar ya que no viene por defecto instalado

Tabla A-0-1 – Comando para instalar PySNMP (Opción1)

```
pip install pysnmp
```

ó

Tabla A-0-2 – Comando para instalar PySNMP (Opción 2)

```
easy_install pysnmp
```

A.3 Instalación de la librería y herramientas para gRPC

Por último, debemos instalar GRPC para ello recurrimos a la web de gRPC [24].

B Registrar MIBs

En este anexo se describirá el procedimiento a seguir para poder registrar distintas MIBs en el sistema.

Para poder registrar una MIB debemos de tener instalada la librería PySNMP, tal como se ha explicado en el anexo A.2. Debemos conocer las dependencias que pueda tener esta MIB con respecto a otras; para ello miraremos en la sección "`IMPORTS`" de la propia MIB y ahí encontraremos las distintas MIBs que se necesitan para poder utilizar la MIB de manera correcta. En la Tabla B-0-3, se detalla un ejemplo de cómo consultar estas dependencias. En este caso, las MIBs adicionales a incorporar serian SNMPv2-SM1 y SNMPv2-TC.

Tabla B-0-3 – Ejemplo de una sección "`IMPORTS`" de una MIB

```
IMPORTS
          MODULE-IDENTITY, OBJECT-TYPE
                      FROM SNMPv2-SMI
          DateAndTime, TruthValue, DisplayString
                  FROM SNMPv2-TC
                           AgentNotifyLevel, dlink-commond-mgmt
                                   FROM DLINK-ID-REC-MIB;
```

Tras conocer las dependencias que tiene la MIB que queremos instalar, utilizaremos una herramienta llamada "`mibdump.py`", que es parte del paquete PySNMP, y se encarga de generar el código Python que permitirá trabajar con los datos en el formato que se define en la MIB. A modo de ejemplo, en la Tabla B-0-4, se detalla el comando que habría que ejecutar para generar el código Python correspondiente con la MIB Equipment de Dlink.

Tabla B-0-4 – Comando mibdump.py

```
mibdump.py SNMPv2-SMI SNMPv2-TC DLINK-ID-REC-MIB.mib EQUIPMENT-MIB
```

C Agregar nuevos samplers al sistema

En este anexo se describe el procedimiento a seguir para agregar nuevos tipos de *Samplers* al sistema. Para facilitar la explicación, utilizaremos el "`Sampler_RAM`" a modo de ejemplo.

C.1 Definir el código del *Sampler*

Para empezar, debemos de crear el nuevo Sampler en el directorio framework/nodecontroller/Samplers; este, debe importar las clases indicadas en la Tabla C-0-5 . En particular, se importa la interfaz "`_Sampler`" (línea 1) que todo *Sampler* debe implementar para ser compatible con el sistema, la clase "`Sample`" que se usa para almacenar una muestra y poder mandarla al *IPFIX Exporter* y la función getbulkram del Módulo GetBulkRAM que sirve para poder realizar las distintas peticiones mediante la operación GetBulk y procesar la respuesta guardándola en un diccionario; en concreto GetBulkRAM se encarga de preguntar por los objetos relacionados con la RAM. El resto de clases son genéricas de Python.

Tabla C-0-5 – Importación de clases del Sampler

```
1   from ._Sampler import _Sampler
2   from .sample import Sample
3   from NodeController.GetBulkRAM import getbulkram
4   from collections import defaultdict
5   from datetime import datetime
```

Para que un *Sampler* funcione, deberemos implementar dos funciones. La primera es la función "`validate()`" y, como se ilustra en la Tabla C-0-6, comprueba si el identificador de plantilla pasado al Sampler corresponde con el de la muestra que creará el *Sampler*. En ese caso devuelve "`None`"; si no fuera así, devuelve una cadena de texto en la que se detalla el error.

Tabla C-0-6 – Función "`validate()`" del *Sampler*

```
1  def validate(self):
2    if(self._templateId != Sampler_RAM.templateId):
3      return("Incorrect templateId should be 502")
4  return(None)
```

La otra función que se deberá implementar es "`do_sampling()`" y se encarga de llamar a la función "`GetBulk_RAM()`"(línea 2 de la Tabla C-0-7); seguidamente, calcula valor de la marca de tiempo de la muestra (línea 3) y procesar la respuesta de la función "`GetBulk_RAM()`",que está en forma de diccionario extrayendo los valores deseados (líneas 5-18).Los valores obtenidos los agregamos a una nueva instancia de la clase "`Sample`" (línea 20) a la que le establecemos también el identificador de dominio de observación (línea 21), la marca de tiempo (línea 22), el identificador de plantilla IPFIX (línea 23) y una lista con los valores de esta muestra (línea 24). Por último, llamamos a la función "`runCallbacks()`" del objeto SNMP para desencadenar el envío de la muestra (línea 25), y reanudamos el muestreo según el periodo de tiempo que se haya configurado al *Sampler* (línea 26):

Tabla C-0-7 – Función "`do_sampling()`" del *Sampler*

```
 1  def do_sampling (self):
 2    dic_RAM = getbulkram(self._objetsnmp.getDeviceIP(),
                           self._objetsnmp.getDevicePort())
 3    delta = datetime.utcnow() - datetime(1970, 1, 1)
 4    valueslist = []
 5    try:
 6      values = {
 7        'observationPointId': self._observationpoint,
 8        'agentDRAMutilizationTotalDRAM':
                           dic_RAM['agentDRAMutilizationTotalDRAM'][1],
 9        'agentDRAMutilizationUsedDRAM':
                           dic_RAM['agentDRAMutilizationUsedDRAM'][1],
10        'agentDRAMutilization': dic_RAM['agentDRAMutilization'][1]
11      }
12    except:
13      values = {
14        'observationPointId': self._observationpoint,
15        'agentDRAMutilizationTotalDRAM': 0,
16        'agentDRAMutilizationUsedDRAM': 0,
17        'agentDRAMutilization': 0
18      }
19    valueslist.append(values)
20    sample = Sample()
21    sample.setObsDomainId(self._observationdomain)
22    sample.setTimeStamp(delta.total_seconds())
23    sample.setTemplateId(Sampler_RAM.templateId)
24    sample.setData(valueslist)
25    self._objetsnmp._runCallbacks(
                             Sampler_RAM.CALLBACK_MESSAGE_RAM,sample)
26    self._scheduleNext()
```

C.2 Definir las entidades y plantillas de IPFIX

Otro elemento que hay que definir para que el *Sampler* funcione es la plantilla IPFIX que este debe usar. Para ello, primero debemos definidas las entidades IPFIX que usará la plantilla. Estas entidades se definen en el fichero Constants.py situdado en el directorio Framework/Protocol/IPFIX.

Tabla C-0-8 – Entidades IPFIX

```
1       ie_pen = {
2        PEN_UAM: {
```

```
3         2000: {'name':'etherStatsOversizePkts', 'type': 'unsigned32'},
4         2001: {'name':'etherStatsFragments',    'type': 'unsigned32'},
5         2002: {'name':'etherStatsJabbers',      'type': 'unsigned32'},
6         2003: {'name':'etherStatsCollisions',   'type': 'unsigned32'},
7         2004: {'name': 'etherStatsPkts64Octets', 'type': 'unsigned32'}
8       }
```

En PEN_UAM agregaremos los campos que deseamos utilizar dándole un identificador aún no utilizado e indicado el tipo.

Por ejemplo, en la línea 3 de la Tabla C-0-8 se define “etherStatsOversizePkts” cuyo identificador es el 2000 y es de tipo “unsigned32”.

Para definir una nueva template IPFIX vamos a Agent/data y abrimos el fichero IPFIX_templates.json

Tabla C-0-9 – Template en IPFIX_templates.json

```
1    "502": {
2       "name": "RAM",
3       "fields": [
4         {"name":"observationPointId","enterprise": "IANA"},
5         {"name":"agentDRAMutilizationTotalDRAM","enterprise": "UAM"},
6         {"name":"agentDRAMutilizationUsedDRAM","enterprise": "UAM"},
7         {"name":"agentDRAMutilization","enterprise": "UAM"}
8        ]
9    },
```

Debemos especificar el identificador de la template, tal como se indica en la línea 1 de la Tabla C-0-9, debemos ponerle un nombre (línea 2) , por último debemos de poner los campos de la template, tal como se indica en las líneas 1-7, usaremos constantes definidas en constants.py.

C.3 Integrar el *Sampler* en el agente

Debemos ir al directorio Framework/nodecontroller/Samplers y abrimos el __init__.py, en el cual importamos el sampler que hemos creado cómo se muestra en la Tabla C-0-10.

Tabla C-0-10 – Importar Sampler en Factoría

```
1   from .Sampler_RAM import Sampler_RAM
```

Y por último, debemos añadir en CLASS_MAPPING la palabra asociada a ese sampler, tal como se muestra en la Tabla C-0-11.

Tabla C-0-11 – Añadir Sampler en Factoría

```
1   CLASS_MAPPING = {
2           'statistics': Sampler_Statistics,
3           'history': Sampler_History,
4           'iftable': Sampler_ifTable,
5           'CPU': Sampler_CPUs,
6           'RAM': Sampler_RAM,
7           'FLASH': Sampler_FLASH
8       }
```

D Agregar nuevos comandos a la terminal

En este anexo hablaremos de como añadir nuevos comandos a la terminal.

Primero debemos ir al directorio Framework/Protocol/GRPC y en IEACP.proto crearemos un servicio, para cualquier duda visitar grpc.io [24].

Ejecutamos generate_proto.sh que generará los ficheros necesarios para que gRPC funcione de manera correcta.

Vamos al directorio Terminal/Commands y creamos un fichero Command_name.py, donde name es el nombre del comando, en el ejemplo de la Tabla D-0-12 el name es *Agent_Create.*

Tabla D-0-12 – Ejemplo Comando Terminal

```
1  from Lib.VTY import _Command
2  class Command_Agent_Create(_Command):
3      def getCommand(self):   return('agent create')
4      def getShortHelp(self): return('Create agent rmon or snmp')
5  …def getLongHelp(self):
         'Create agent rmon or snmp, necessary: (nameagent,typeagent (rmon or
   snmp),ip,port,mpModel).\n' +'    agent create')

6  def run(self, *args):
7  if(len(args) < 4): return(self.getLongHelp())
8          argumentos = list(args)
9           self.getHandler().agent_create(argumentos[0],argumentos[1],argumento
   s[2],argumentos[3])
10         return('Create agents rmon or snmp request send.')
```

Primero debemos de poner el nombre de la clase acorde al comando que se desea implementar, en este caso en la línea 2 de la Tabla D-0-12 el nombre es "Command_Agent_Create" por lo que implementa un comando para crear un agente. Tras esto debemos de definir el comando que se deberá escribir en la terminal (línea 3), una descipción corta (línea 4), una descripción larga (línea 5). Después, se debe definir una función "def run(self,*args)" que será la que se encargue de pasar los parámetros necesarios a la función, que se deberá de crear en Client.py situado en Terminal y desde esta se llamará a la función concreta del agentStub que le pasará el mensaje pertinente al NodeController (línea 6 -10). Comentar que en la línea 7 se realiza una comprobación del número de argumentos y en caso de que sean menor de los necesarios, en terminal saldrá la ayuda del comando.

Tras crear el comando debemos ir a Main.py situado en Terminal, donde deberemos importar el comando y añadirlo a la VTY, tal como muestra laTabla D-0-13.

Tabla D-0-13 – Añadir comandos

```
vty.addCommand(el comando importado)
```

Tras esto nos vamos a Agent y en AgentServicerImpl.py deberemos crear la función que recibirá los parámetros por gRPC y llamará a la función del manager correspondiente. Lógicamente el nuevo comando deberá realizar tareas para las que el manager, en un principio, no esta diseñado, por lo que es de esperar que sea necesario crear nuevas funciones en el manager y en los módulos inferiores.

E MIBs utilizadas

En este anexo hablaremos de las MIBs utilizadas.

Tabla E-0-14 – MIBs utilizadas

Objeto	MIB
etherStatsOversizePkts	RMON-MIB (estandarizado)
etherStatsFragments	RMON-MIB (estandarizado)
etherStatsJabbers	RMON-MIB (estandarizado)
etherStatsCollisions	RMON-MIB (estandarizado)
etherStatsPkts64Octets	RMON-MIB (estandarizado)
etherStatsPkts65to127Octets	RMON-MIB (estandarizado)
etherStatsPkts128to255Octets	RMON-MIB (estandarizado)
etherStatsPkts256to511Octets	RMON-MIB (estandarizado)
etherStatsPkts512to1023Octets	RMON-MIB (estandarizado)
etherStatsPkts1024to1518Octets	RMON-MIB (estandarizado)
etherStatsDataSource	RMON-MIB (estandarizado)
etherStatsOwner	RMON-MIB (estandarizado)
etherStatsStatus	RMON-MIB (estandarizado)
etherStatsDropEvents	RMON-MIB (estandarizado)
etherStatsOctets	RMON-MIB (estandarizado)
etherStatsPkts	RMON-MIB (estandarizado)
etherStatsBroadcastPkts	RMON-MIB (estandarizado)
etherStatsMulticastPkts	RMON-MIB (estandarizado)
etherStatsCRCAlignErrors	RMON-MIB (estandarizado)
etherStatsUndersizePkts	RMON-MIB (estandarizado)
agentDRAMutilizationTotalDRAM	AGENT-GENERAL-MIB (propietario de Dlink)
agentDRAMutilizationUsedDRAM	AGENT-GENERAL-MIB (propietario de Dlink)
agentDRAMutilizationUnitID	AGENT-GENERAL-MIB (propietario de Dlink)
agentCPUutilizationIn5sec	AGENT-GENERAL-MIB (propietario de Dlink)
agentCPUutilizationIn1min	AGENT-GENERAL-MIB (propietario de Dlink)
agentCPUutilizationIn5min	AGENT-GENERAL-MIB (propietario de Dlink)
agentFLASHutilizationTotalFLASH	AGENT-GENERAL-MIB (propietario de Dlink)
agentFLASHutilizationUsedFLASH	AGENT-GENERAL-MIB (propietario de Dlink)
agentFLASHutilization	AGENT-GENERAL-MIB (propietario de Dlink)
ifInOctets	IF-MIB (estandarizado)

ifInUcastPkts	IF-MIB (estandarizado)
ifInNUcastPkts	IF-MIB (estandarizado)
ifInDiscards	IF-MIB (estandarizado)
ifInErrors	IF-MIB (estandarizado)
ifInUnknownProtos	IF-MIB (estandarizado)
IfOutOctets	IF-MIB (estandarizado)
ifOutUcastPkts	IF-MIB (estandarizado)
ifOutNUcastPkts	IF-MIB (estandarizado)
ifOutDiscards	IF-MIB (estandarizado)
ifDescr	IF-MIB (estandarizado)
ifOutErrors	IF-MIB (estandarizado)
ifOutQLen	IF-MIB (estandarizado)
ifSpecific	IF-MIB (estandarizado)
ifType	IF-MIB (estandarizado)
ifMtu	IF-MIB (estandarizado)
ifSpeed	IF-MIB (estandarizado)
ifPhysAddress	IF-MIB (estandarizado)
ifAdminStatus	IF-MIB (estandarizado)
ifOperStatus	IF-MIB (estandarizado)
ifLastChange	IF-MIB (estandarizado)